Writing an X compositing manager

Arnaud Fontaine (08090091)

31 August 2009

Contents

List of figures

List of listings

List of Tables

Introduction and rationale

This project is about writing a free software *compositing manager* (Wikipedia 2009-02-25) on top of the X WINDOW PROTOCOL (Wikipedia 2009-02-24). Basically, a compositing manager is a piece of software running along with the window manager and where each graphical program outputs into a separate and independent off-screen buffer that can be manipulated before being shown in order to enhance user experience. Unlike a *compositing window manager*, a compositing manager does not manage windows but simply implements visual effects such as windows translucency, drop shadows, fading... Nowadays, major operating systems provide such technologies, for instance `Quartz` compositor (Wikipedia 2009-01-28, Siracusa 2005-04-28, Apple 2008-10-15) on `Mac OS X`, `CompizFusion` (Wikipedia 2009-03-09*a*) on `Unix`-like operating systems, or `DWM` (*Desktop Window Manager* (Wikipedia 2009-02-19)) on Windows Vista.

More precisely, this project aims to write an efficient, lightweight and responsive compositing manager in C programming language, based on the XCB [1] (*X protocol C-language Binding*) client library for the X WINDOW PROTOCOL. In addition, it will be based on ICCCM (Standard 1993-12) and EWMH (Group 2006-09-29) specifications to communicate with window managers in a standardized way, therefore minimizing modifications in existing window managers as they usually already implement these specifications. XCB is available on any operating systems using the X WINDOW SYSTEM (usually XORG [2] implementation), consequently this project will benefit to any of those systems and existing EWMH-compliant window managers. This project also aims to provide a functional and stable software to improve window manager usability from an end-user point of view (for instance windows translucency,

[1] `http://xcb.freedesktop.org/`

[2] `http://www.x.org/wiki/`

application-switcher using live thumbnails instead of plain icons, EXPOSÉ (Wikipedia 2009-03-09*b*)...), therefore it does not intend at all to provide useless eye-candy effects. Finally, it would also benefit the XCB project because I will use XCB implementations of X extensions which has not been extensively tested yet.

Currently, besides of compositing window managers such as `Compiz` [3], the only alternatives with the same purpose as this project are `xcompmgr` [4] and `cairo-compmgr` [5]. However, the former is a proof-of-concept based on a monolithic architecture which only implements basic features using XLIB whereas the latter uses Cairo [6] graphic library whose drawbacks for this project, beside being heavy, are described in section A.3.

I am really interested in X WINDOW SYSTEM, so this project would give me the opportunity to write a program from scratch to learn more about X programming. Moreover, I have planned to use it on a daily basis because I have always wanted a compositing manager fitting my needs.

Firstly, I will introduce the X WINDOW SYSTEM and especially the mechanisms relevant to my project. This chapter will also get into details about XCB and explain why it has been chosen over the other X client library, namely XLIB. This project is divided in three main parts, namely the core code, the rendering backend and the effect plugins. Therefore, secondly, I will describe the implementation of the core code, including X extensions involved, and its issues and shortcoming along with ideas to address them, finally this chapter will describe ideas about future features. Thirdly, I will describe the rendering backends architecture, currently based on RENDER X extension outlined along with hardware acceleration mechanisms, which will be extended in the future to support other rendering methods such as OPENGL through libraries detailed in this chapter. Fourthly, I will introduce effect plugins and their general architecture, then the plugins implemented, currently OPACITY and EXPOSÉ, finally this chapter will give current issues and ideas about future work, including possible implementations of plugins I unfortunately lacked time to implement.

[3] `http://compiz-fusion.org/`
[4] `http://freedesktop.org/wiki/Software/xapps`
[5] `http://cairo-compmgr.tuxfamily.org/`
[6] `http://cairographics.org/`

Acknowledgement

I woud like to thank Vincent Torri and Julien Danjou for their patience when answering to my questions about X WINDOW PROTOCOL and more generally about their advices while writing the code, and of course Faye Mitchell for supervision and guidances throughout this project.

Chapter 1

X WINDOW SYSTEM

In this document, all* X *requests, replies, events and errors mentioned comes from the* X WINDOW PROTOCOL *(X Consortium Standard 2004) rather than the* X *client library.

1.1 X WINDOW PROTOCOL

1.1.1 Introduction

The X WINDOW SYSTEM (commonly X11 or X) is a standard and complete network-transparent graphical windowing system based on a client/server model built on top of an operating system kernel.

A given client and the server communicate asynchronously via the X WINDOW PROTOCOL (X Consortium Standard 2004) which specifies that *the protocol is intended to provide mechanisms, not policies*, therefore this protocol does not specify:

- Inter-clients interactions to transfer data between windows (selections, cut and paste and drag and drop are some common examples) but also between the window manager and the windows it manages. These interactions are however described in specifications like ICCCM (Standard 1993-12) and EWMH (Group 2006-09-29);
- Common widgets like buttons, menus, textbox... but toolkits can be built on top of this

protocol in order to provide these features.

Unlike traditional client/server applications, the server generally runs on the local machines and owns the input devices, allowing the remote or local client programs to draw or display fonts on the display by sending requests.

Usually, there is at least one client, known as a **window manager**, which interacts with the X server in order to manage the windows on a screen. There may have another client, a **compositing manager** (Wikipedia 2009-02-25) which composites and paints windows in an off-screen buffer before displaying them on the display. The figure 1.1 shows a window manager over an UNIX domain socket managing windows from the root window on an off-screen buffer and a compositing manager over TCP/IP which actually paints on the screen, but ignores completely `InputOnly` windows as these windows are never visible.

The following types of messages may be exchanged over the wire: **request**, **reply**, **event** and **error**.

The X WINDOW PROTOCOL messages given above may be extended through extensions described in further details in the next chapters.

1.1.2 Identifiers of resources

Resources such as windows, pixmaps and graphic contexts are stored server-side and destroyed by default when the server is reset. When the client requests to create a new resource, the following steps are executed:

1. The client asks, through a client library usually, the current identifier (XID) range to the server and commonly picks the next sequential number;
2. The server allocates the resource and associates it to the given XID;
3. The client can now perform operations on this resource by specifying this identifier.

This mechanism avoids copy of the resource between the server and the client which requested the resource allocation and also enables resources sharing among clients on the same X SERVER, consequently resulting in a more efficient use of the network.

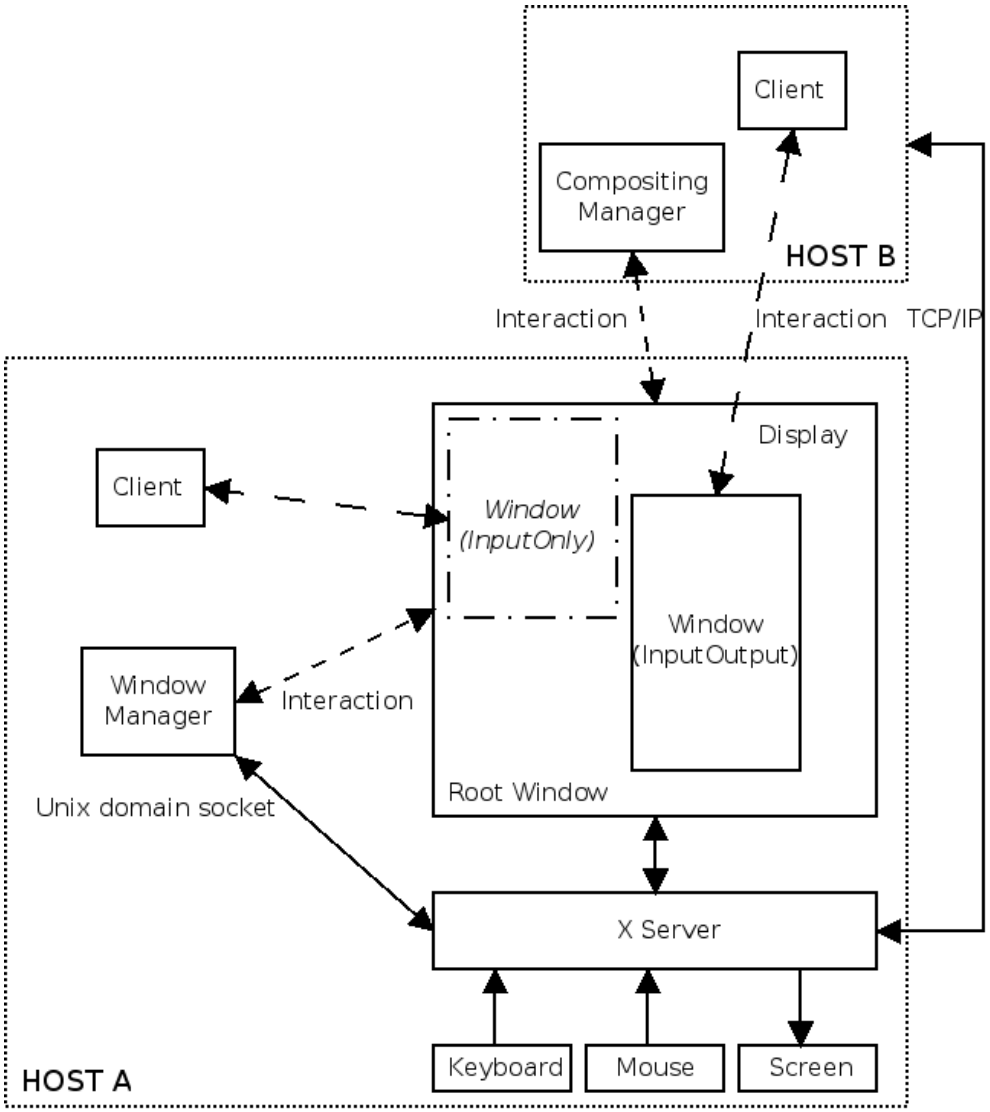

Figure 1.1: *A compositing manager and window manager using the same display at the same time with different protocols*

1.1.3 `Atom`

An `Atom` is a an unique 32 bits integer identifier assigned by the server (as opposed to a resource identifier explained in section 1.1.2) used often in inter-clients communications. As the identifier has a fixed and short length, it is consequently faster to process on a network. The string is stored in the server and referenced as its name.

A client can request allocation of an `Atom` by sending the intended name in a `InternAtom` request, the server replies by sending its identifier. It is automatically created if it does not already exist. The opposite operation is achieved by sending `GetAtomName` request.

Contrary to the default behavior of a resource identifier, an `Atom` is kept even when the server is reset. In order to reduce network usage, most common `Atoms` needed by applications are already allocated in the server, such as most ICCCM atoms (`WM_CLASS` and `WM_NAME` for example).

1.1.4 `Window`

In the X WINDOW SYSTEM, the windows are hierarchically ordered as a tree where the **root window** is the parent of all other windows (or more precisely known as *subwindows*). This *special* window is generally as large as the physical screen and is situated behind all its children. A window is considered as a **`Drawable`** like `Pixmap` explained later.

When the client sends a request to create a window (namely `CreateWindow` in the X WINDOW PROTOCOL and `DestroyWindow` to destroy it) or more precisely a subwindow of an existing window, it needs to specify at least the following:

- Identifier (XID) of its parent;

  A top-level window (e.g. a child of the root window) is created by setting the parent window to the identifier of the root window. But as other screen information, the identifier of the root window is sent by the server when the connection has been successfully established.

- Class of the window;

  - `InputOnly`;

    Specify that the window can receive events but cannot be used as a source or destination for graphics requests, therefore it is never painted on the screen by a compositing manager nor a window manager.

  - `InputOutput`;

    Specify that the window can receive events and can be used for drawing.

A window is not visible on the screen before being *mapped* by issuing a `MapWindow` request (`UnmapWindow` to *unmap* it). It is worth noticing that the window content is not guaranteed to be preserved while the window is *unmapped*, therefore, it does not make sense to draw anything into an *unmapped* window.

A window has attributes (map state, border, cursor, accepted events...) stored server-side. They can be fetched and set respectively by sending `GetWindowAttributes` and `ChangeWindowAttributes` requests. Window decorations are accomplished by the window manager, not the client itself, usually by creating a separate window for the titlebar and then re-parenting the original window to this one (explained in section 2.4).

A window also has properties allowing for example to inform the window manager about the behavior it desires (the figure 1.1 shows some common properties). A property is stored server-side as an `Atom` and thus characterized by its name, type and value. It can be fetched or set by sending respectively `GetWindowProperty` and `ChangeWindowProperty` request. By design, the X WINDOW PROTOCOL does not describe these properties, rather defined in ICCCM and EWMH specifications, but it however provides the necessary requests and reply formats to manage them. For instance, EWMH specifies that the window title is stored in `_NET_WM_NAME` window property as an UTF-8 string.

The window geometry can be obtained by a `GetGeometry` and set by a `ConfigureWindow` request. It is worth mentioning that the geometry does not include the border also returned by the former request.

```
WM_STATE(WM_STATE):
                window state: Normal
                icon window: 0x0
_AWESOME_PROPERTIES(STRING) = "100000000"
_NET_WM_PID(CARDINAL) = 12313
WM_PROTOCOLS(ATOM): protocols  WM_DELETE_WINDOW, _NET_WM_PING
WM_LOCALE_NAME(STRING) = "fr_FR.UTF-8"
WM_CLASS(STRING) = "urxvt", "URxvt"
WM_HINTS(WM_HINTS):
                Client accepts input or input focus: True
                Initial state is Normal State.
                window id # of group leader: 0x2600007
WM_NORMAL_HINTS(WM_SIZE_HINTS):
                program specified minimum size: 11 by 18
                program specified resize increment: 7 by 14
                program specified base size: 4 by 4
                window gravity: NorthWest
```

```
WM_CLIENT_MACHINE(STRING) = "maggie"
WM_COMMAND(STRING) = { "urxvt" }
_NET_WM_ICON_NAME(UTF8_STRING) = 0x61, 0x72, 0x6e, 0x61, 0x75, 0x40,
  0x6d, 0x61, 0x67, 0x67, 0x69, 0x65, 0x3a, 0x20, 0x7e
WM_ICON_NAME(STRING) = "arnau@maggie: ~"
_NET_WM_NAME(UTF8_STRING) = 0x61, 0x72, 0x6e, 0x61, 0x75, 0x40, 0x6d,
  0x61, 0x67, 0x67, 0x69, 0x65, 0x3a, 0x20, 0x7e
WM_NAME(STRING) = "arnau@maggie: ~"
```

Listing 1.1: *Window properties defined by* `urxvt` X *terminal*

1.1.5 `Pixmap`

A `Pixmap` is just a tree dimensional array of bits used for drawing and is therefore considered as a `Drawable` (like a window). A `Pixmap` is an off-screen resource which can be partially or completely transferred to a window and vice-versa. It is often used as a picture buffer or a background pattern. In the X WINDOW PROTOCOL, a client requests allocation of a `Pixmap` using `CreatePixmap` and can be freed by `FreePixmap`.

1.1.6 Events

According to the X WINDOW PROTOCOL specification, clients are informed by means of events and can be generated from input/output devices owned by the X SERVER, or also as side effects of clients requests (for instance by a `SendEvent` request which allows a client to send a request to another client). As described in section 1.1.1, events is one of the four specific messages exchanged between the server and the clients. Each event message usually specifies its own format.

An event is relative to a window. In order to receive specific events, a client has to explicitly state what events it is interested in concerning a given window. This can be achieved by specifying a *mask* when creating the window by sending a `CreateWindow` request or at anytime thanks to `ChangeWindowAttributes` request.

For instance, a client may set `EventMask` to `KeyPress` and `Expose` allowing a window to received these events from the server when a key a button has been pressed or when the window content is invalid meaning that the window has to be redrawn (as explained in section

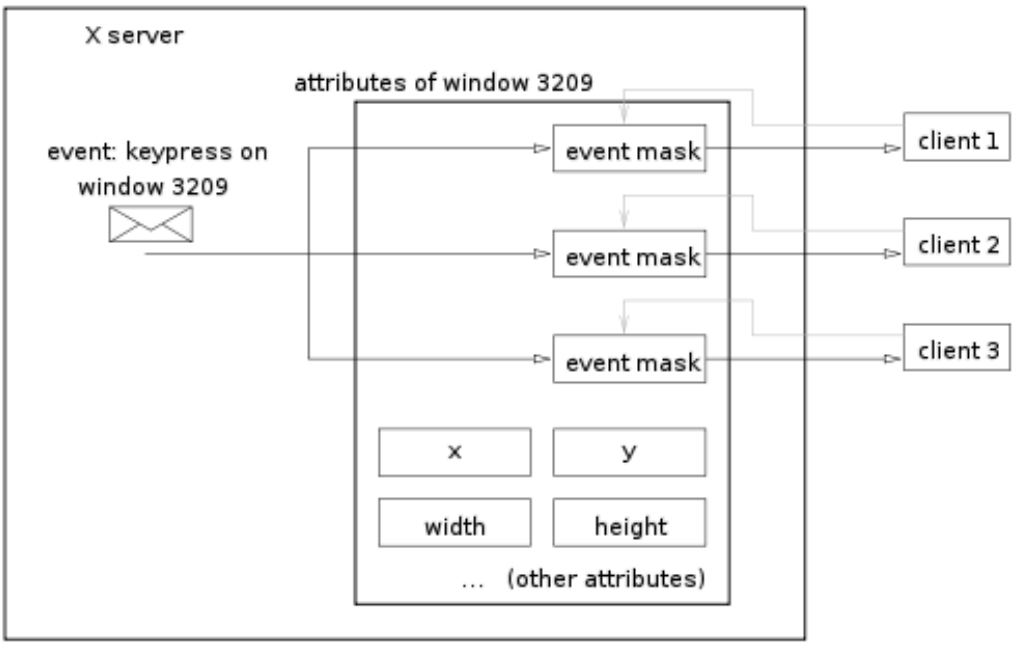

Figure 1.2: *Generated* `KeyPress` *event (source: Wikipedia)*

1.1.4).

1.1.7 Keyboard and pointer

Each physical or logical keyboard key is associated with an unique key code (known as a `KeyCode` in the X WINDOW PROTOCOL specification). It may be associated to at least one character or special key, names a `KeySym`, selected thanks to special keys called modifiers. In most common case, the following modifiers are available on the end-user keyboard: `Shift`, `Control`, `Meta`, `Compose`, `CapsLock` and `NumLock`.

The association table between `KeyCodes` and `KeySyms` are maintained server-side making it available and modifiable to all clients. It allows the end-user to use his own keyboard layout.

Within a window, when the key state changes or the pointer moves, the following (self-explanatory) event messages may be generated by the server:

- keyboard related events:

  - `KeyPress`;
  - `KeyRelease`.

- pointer related events:
  - `ButtonPress`;
  - `ButtonRelease`;
  - `MotionNotify`.

All these event messages share the same format whose fields are the root window, source window and coordinates available for both. Depending on the source of the event, it also contains a *detail* field holding the key code or the button and a *state* field storing the current keyboard modifiers or mouse buttons. The server does not convert a `KeyCode` associated with modifiers `KeySyms` in a `KeyCodes` or vice versa, it is actually performed by the client itself.

For a given window, a client can grab or ungrab (e.g. consequently all events will respectively be sent or not to this client) the following:

- A key ((`Un`)`GrabKey` request);
- The pointer ((`Un`)`GrabPointer` request);
- The keyboard ((`Un`)`GrabKeyboard` request).

1.1.8 `Visual`

A **`Visual`** (consortium 2005-10-31) contains information about the possible color mapping dependent on display hardware. There are several visual classes but the most common one nowadays is `TrueColor` where a 24 bits pixel value is decomposed into separate RGB values where each channel is 8 bits long. An example of available `Visuals` are shown in section A.8.

1.2 X client library

A client program can interact with the X SERVER using a X client library implementing the X WINDOW PROTOCOL, like XLIB and XCB. This section will briefly describes both libraries and also explain why XCB has finally been chosen over XLIB.

1.2.1 XLIB

1.2.1.1 Overview

The XLIB is a protocol X client library written in C programming language started in 1985 and currently supported by any operating systems relying on the X WINDOW SYSTEM, mostly UNIX-like operating systems. This library is a free software (under the terms of the MIT license) based on a monolithic architecture implementing ICCCM standard. Currently, almost every widget toolkits are built on top of this library to communicate with the X SERVER. It provides specific inspection and management operations on the client-side event queue and defines its own types for (almost) everything.

Requests to the server: In most common usage and cases, when a request is sent with XLIB, it simply sends the request and blocks until the reply is received, thus making the request and reply synchronous, because the XLIB asynchronous way is well-known for not being thread-safe and complicated to use.

1.2.1.2 Pros

- Well established;
- Well documented.

1.2.1.3 Cons

- Monolithic architecture resulting in a big library size;

- Complex and ugly code;
- Inconsistent API;
- Requests requiring a reply are synchronous;
- Not thread-safe.

1.2.2 XCB

1.2.2.1 Overview

XCB [1] (X C BINDING) is a C-language binding for the X WINDOW SYSTEM. It is a replacement for XLIB *featuring a small footprint, latency hiding, direct access to the protocol, improved threading support and extensibility* (quoted from official XCB website description). The core and extension X WINDOW PROTOCOL are described as XML files generated in C programming language via XSLT. Like XLIB, it is a free software licensed under the terms of the MIT license. Barton Massey and Robert Bauer proved key portions of XCB formally correct using Z notation. Unlike XLIB, XCB is based on a modular architecture.

Like XLIB, XCB provides types for resources except that XCB relies on fixed-size integers data type standardized by ISO C99 standard (commonly available in `stdint.h` C library header). Instead of defining macros, XCB makes extensive usage of `enums`, thus avoiding to define specific structures uselessly.

Unlike XLIB functions, XCB does not provide a way to inspect and manage the event queue. Indeed, once an event or an error is dequeued, it cannot be requeued afterwards, meaning that the client cannot step through items in the queue (this kind of operations adds much more complexity and are not really needed, especially within a low-level library like XCB).

Requests to the server: When a request is sent with XCB, it returns a *cookie* which is an identifier used to get the reply at anytime. The type of cookie is `xcb_void_cookie_t` for requests that do not generate a reply from the server, whereas for other requests, each one has its

[1] `http://xcb.freedesktop.org`

own cookie type, for instance `xcb_get_window_attributes_cookie_t` is returned by `xcb_get_window_attributes()` (corresponding to `GetWindowAttributes` request in the X WINDOW PROTOCOL). It is preferable to send the request and then ask for the reply as later as possible to ensure it has already been processed by the X SERVER. Therefore, This mechanism makes requests and replies completely asynchronous.

XCB provides two modes, namely *checked* and *unchecked*, defining where the request is stored for further processing by the client. If a request generates a protocol error in the former mode, then the error goes into the event queue and may be processed in the event loop, whereas the error is held aside in the latter mode until the client asks explicitely for the reply. An error is just a structure names `xcb_generic_error_t` where the `response_type` field equals to 0.

1.2.2.2 XCB-UTIL library

XCB intends to be a lower-level library than XLIB, as such it does not provide facilities such as ICCCM helperss or event handler facilities, which is instead implemented in a modular way as part of an independent library, XCB-UTIL (described in further details in section A.2.1).

1.2.2.3 Pros

- Modular architecture resulting in smaller library size;
- Making multithreading easier and reliable;
- Consistent API;
- Completely asynchronous.

1.2.2.4 Cons

- Sometimes a bit tricky to make the program fully asynchronous;
- Not well documented;
- Extensions missing (XKB ...).

The cons points quoted above result from youthfulness of the XCB project. In addition, most functions are easily understandable by reading XLIB documentation or the X WINDOW PROTOCOL specification which are quite complete.

1.2.3 XLIB/XCB round-trip performance comparison

The listing A.4 given in section A.7 shows an example program comparing XLIB and XCB performances about requests and replies (also known as a round-trip). It sends 500 atoms and displays the following output:

```
=> XCB wrong usage (synchronous)
Duration: 0.038608s

=> XCB good usage (asynchronous)
Duration: 0.002691s
Ratio   : 14.35

=> Xlib traditional usage
Duration : 0.043487s
Ratio    : 16.16
```

Listing 1.2: *Output of program given in section A.7*

This output clearly shows a dramatic speedup of requests and replies of asynchronous XCB method compared to XLIB traditional synchronous way of sending a request and blocks until the reply is ready. In this case, XCB is about 16 times quicker than XLIB. This difference becomes more and more obvious as the number of `Atoms` requests sent following this method increases. XCB also provides a slight speedup compared to XLIB even when both of them use the same method.

1.2.4 EWMH XCB implementation

This project will heavily used EWMH to communicate with the window manager to minimize modifications of existing window managers. For instance, it allows the window manager to declare the properties it supports through `_NET_SUPPORTED` root window property or also the windows to set the name displayed through `_NET_WM_NAME` property. This specification

lies on top of ICCCM, for instance concerning an important inter-clients mechanism, namely ownership meaningful to exchange data between clients.

There is already an implementation of ICCCM as XCB-UTIL/icccm, however there is no such implementation for EWMH (even XLIB does not provide one). Therefore, I have implemented an xcb-ewmh library for XCB as part of this project (described in further details in section A.2.1.3).

Chapter 2

Core code

This chapter describes the core code and more generally the overall architecture of the compositing manager. It also describes extensions used in the core code.

2.1 Overview

The code is split up in three main components: the **core code**, **rendering backends** (chapter 3) and **effect plugins** (chapter 4). The decision to split the code has been leaded by technical reasons and also to attract contributors as explained in chapter 4 rationale. Besides startup routines and configuration file parsing (described in section A.6.2), the core code aims to bind together the rendering backend and the effect plugins but also to provide helpers functions used by all of them. This code aims to be as small as possible and let the rendering backend and effect plugins handle more complex and specific tasks. The source code lies on top of several libraries (mainly XCB-UTIL libraries however) listed in section A.2.

2.2 X extensions

Before getting into details about the core code, this section describes the X extensions particularly relevant when writing a compositing manager (Höglund n.d., Keith Packard 2000-06-

15, Fedora 2008-05-24), but it does not include the extension used for the rendering backend (as described in section 3.3).

2.2.1 COMPOSITE

This extension places the pixel contents of a selected windows hierarchy to an off-screen buffer allowing windows manipulation before being shown (Keith Packard 2007-07-03). `Composite` extension is usually used along with a `Render`-like extension to perform compositing in an off-screen buffer and then to update the screen.

The pixels in the off-screen buffer for a previously selected windows hierarchy are only available when visible (e.g. when the window is mapped) and automatically reallocated when the size of the top-level window changes. The selection mechanism is completely different from `Mac OS X` implementation (Siracusa 2005-04-28, Apple 2008-10-15) because the latter has top-level windows only (Keith Packard 2000-06-15) unlike the tree organization of windows defined by the X WINDOW PROTOCOL (X Consortium Standard 2004).

The off-screen buffer has the following properties:

- Only contains contents within parent window geometry;
- Automatically reallocated when the top-level window changes its size (relevant to the property explained before);
- Include both window content and borders as well as the content of all its children.

Among the requests provided by this extension, the following ones are useful when writing a compositing manager and actually used in `xcompmgr` and `Compiz` (Wikipedia 2009-03-09*a*) too:

RedirectSubwindows(Window window, UPDATETYPE update)	
Explanation	Redirect the hierarchy of current and future children starting from `window`. `update` may be either set to `Automatic` or `Manual` meaning `window` is automatically updated or not on the screen, when one of the children content is modified in the off-screen buffer.
Compositing Manager	Set `window` to the root window and `update` to `Manual` because it commonly performs several distinct operations on the window contents at the same time before updating the screen, such as blending together the contents of overlay windows or also drawing shadows around them. It may detect when to update the off-screen thanks to events generated by DAMAGE extension as explained in further details in section 2.2.3.
NameWindowPixmap(Window window) => Pixmap pixmap	
Explanation	Only supported from version 0.2 and makes `pixmap` a reference to `window` stored in the off-screen buffer which stores the window contents (namely its borders and the contents of all its descendants). `pixmap` is automatically allocated when the window is mapped or resized. This is the programmer responsibility to free the `Pixmap` (using `FreePixmap` request) once it is not used anymore.
Compositing Manager	As the off-screen buffer only stores the pixels of *mapped* windows, it allows to hold window contents even after being *unmapped*. For example, it is meaningful when animating the disappearance of a window which is going to be destroyed. Moreover, it provides a convenient way to get the `Pixmap` of the actual window content, including the borders and the window decorations (meaningful for re-parenting window managers explained in section 2.4), on server-side, thus saving a lot of requests to achieve the same goal from client-side.

Table 2.1: COMPOSITE X extension requests

Besides the above requests meaningful for RENDER, OPENGL compositing managers may also use `CompositeGetOverlayWindow` and `CompositeReleaseOverlayWindow` for the GLX overlay window (see section 3.5 for details about the future OPENGL backend).

2.2.2 XFIXES

This extension provides server-side support of workarounds for shortcomings in the core X WINDOW SYSTEM (Packard 2006-12-14). This section will only get onto the part relevant for a compositing manager, namely `Region`, supported from version 2.0.

A region is uniquely identified by its XID and represents a *set of disjoint (non overlapping) rectangles plus, an "extent" rectangle which is the smallest single rectangle that contains all the non-overlapping rectangles* (X SERVER code [1]). This extension only exposes this object already used extensively in the X SERVER and needed for `DAMAGE` extension (see section 2.2.3).

Among the requests provided by this extension, the following ones are the most important and useful when writing a compositing manager (actually used in `xcompmgr` and `Compiz`):

CreateRegion(LISTofRECTANGLE rects) => Region region	
Explanation	Create a region initialized to the union of `rects`.
Compositing Manager	A compositing manager can use this request together with `DAMAGE` extension to subtract a certain region from a *damaged* area.
UnionRegion, SubtractRegion(Region source1, Region source2, Region destination)	
Explanation	Combine `source1` and `source2` regions into `destination` region.
Compositing Manager	When used with DAMAGE extension, useful to combine the overall region previously *damaged* with the one currently *damaged*.
SetPictureClipRegion(Picture picture, INT16 clip-x-origin, INT16 clip-y-origin, Region region)	
Explanation	Change `clip-mask` of `picture` to `region` and sets the clip origin to $(clip-x-origin, clip-y-origin)$. The `clip-mask` intends to restrict write of pixels on the `picture` from the clip origin because only bits set to 1 are drawn.
Compositing Manager	Along with DAMAGE extension, it allows to copy only pixels which have been *damaged*, thus avoiding to repaint entirely the screen on each update.

Table 2.2: XFIXES X extension requests

Once a region is not used anymore, it has to be destroyed by issuing a `DestroyRegion`

[1] `http://cgit.freedesktop.org/xorg/xserver/tree/mi/miregion.c#n106`

request to free the memory allocated on server-side.

2.2.3 DAMAGE

This extension is implemented server-side for efficiency reasons and allows to track modified regions (or rectangles) of `Drawables` (either a `Pixmap` or a `Window`) content, by sending these regions as a stream of events, namely `DamageNotifys` (Keith Packard 2007-01-08).

A `Damage` object holds any accumulated monitored `Region` which has been changed (referred as *damaged* in DAMAGE specification document) and *reflects both drawing within the window itself **as well as** drawing within any inferior window that affects pixels seen by `IncludeInferiors` rendering operations* (Keith Packard 2007-01-08) where `IncludeInferiors` is defined in RENDER extension (see section 3.3). For example, if two top-level windows are monitored for changes and one overlays the other, two `DamageNotify` notifications containing the modified areas will be sent.

Among the requests provided by this extension, the following one is the most useful when writing a compositing manager (actually used in `xcompmgr` and `Compiz`):

DamageCreate(Drawable drawable, DamageReportLevel level) => DAMAGE damage	
Explanation	Create `damage` to track changes on `drawable`. `level` specifies the area relative to `drawable` for which `DamageNotify` events will be generated.
Compositing Manager	After creating a new DAMAGE object for each window, it processes the `DamageNotify` events in order to update the off-screen buffer accordingly (as provided by COMPOSITE extension described in section 2.2.1), and then paints the window thanks to X RENDER extension for example. Concerning translucency implementation for example, each time a window is moved around the screen, `DamageNotify` notifications are sent to all *damaged* windows. Each notification contains the area involved by the change and then the alpha blending will be performed on this area to provide *real transparency* (alpha compositing is defined in section A.1).
DamageSubtract(DAMAGE damage, Region repair, Region parts)	
Continued on next page	

Explanation	Modify `repair` and `parts` according to their values and `damage` area and allows for example to subtract all the damage in order to *repair* a window.
Compositing Manager	Meaningful to store a `Region` (thanks to the XFIXES extension described in section 2.2.2) including only the *damaged* area on the screen, thus improving performances by avoiding useless copy of pixels on the screen.

Table 2.3: DAMAGE X extension requests

2.3 Implementation

2.3.1 Architecture

The compositing manager communicates with the window manager and its clients to achieve the desired effects through atoms defined in EWMH specification (Group 2006-09-29). In some cases, it may also rely on ICCCM specification (Standard 1993-12).

At the moment, the list of windows and plugins are implemented as linked-lists, firstly because the size of these lists are modified on runtime and secondly because as linked-list algorithms are trivial to implement it allowed me to focus on the main features.

All public functions and macros of a given file are prefixed by the filename without its extension for readability purpose, that is why the filename is not given for each function call given below.

Whenever possible and relevant, asynchronous requests (e.g. requests generating replies) are split up in two parts on purpose to take full advantage of XCB asynchronous model (the second function is suffixed by `finalise` keyword), this way it allows to send a request and then execute some code before actually getting the reply to avoid blocking like most XLIB program do.

The following sections will get into details about the main parts of the core code, namely the startup process, windows management and events handling. The list of project source code is given in section A.6.

2.3.2 Startup

The compositing manager may be started at any time after the window manager. On compositing manager startup, the following steps will be performed and are divided in several round-trips to decrease latency (`display.c` implements startup routines related to the core code).

First round-trip

1. Call `atoms_init()` to send requests to get the EWMH atoms XID by calling `xcb_ewmh_init_atoms_list()` function from `xcb-ewmh` XCB-UTIL library. Besides EWMH atoms requests, `atoms_init()` also sends requests to get the atoms XID related to the root background `Pixmap` (conventionally stored in `_XROOTPMAP_ID` or `_XSETROOT_ID` root window properties) and opacity atom XID (`_NET_WM_WINDOW_OPACITY` stored as a window property);

2. Prefetch the extension data into the extension cache by calling `xcb_prefetch_extension_data()` function which avoids blocking when actually calling `xcb_get_extension_data()` function;

3. Initialize startup error and event handler by calling `event_init_start_handlers()` function. The former handler exits the program if any request fails as all the requests during startup are required to make the program running properly. The latter sets an handler for `PropertyNotify` event whose purpose is explained in one of the next round-trip;

4. Load the rendering backend in the program address space by calling `rendering_load()` which basically performs a `dlopen()` call on the rendering backend given in the configuration file, then it looks for the `rendering_functions` symbol to get the rendering backend virtual table (further details are given in section 3.2);

5. Process the replies of requests sent previously by `atoms_init()` by calling `atoms_init_finalise()`. The `Atom` XIDs are stored as global variables as they will not change until the X SERVER is reset. It also sends a request to get `WM_SUPPORTED` root window property meaningful for effect plugins to check whether the atoms required are actually supported by the window manager;

6. Call `xcb_ewmh_get_wm_cm_owner()` function which represents the first step toward getting ownership on `_NET_WM_CM_S`n root window property where n is the screen number (this ownership mechanism is described in ICCCM manual and briefly in section 1.2.4). This step is a requirement from the EWMH specification.

Second round-trip

1. First step to initialise extensions, which consists in getting the extensions information stored in the cache initialized in the previous round-trip, by calling `display_init_extensions()` function . It allows to check whether the COMPOSITE, XFIXES and DAMAGE extensions are present on the server-side (indeed the presence of these X extensions on the client-side has already been checked on source code compilation). This function also sends `QueryVersion` requests for each extension, otherwise the behavior of the server is undefined;

2. Initialise the rendering backend by calling its `init()` function (see section 3.2 for further details);

3. Second step for `_NET_WM_CM_S`n ownership which consists in getting the reply of the request sent during the previous round-trip. If the reply value is a window, then the program exits with an error message;

4. First step of compositing manager registration. Firstly, it creates an `InputOnly` window with `override-redirect` window attribute set to `true`. This window is used to take ownership on `_NET_CM_S`n root window property and its XID is stored as a global variable firstly to destroy it on program exit. Secondly, it sends a `ChangeProperty` request on the window name (as stored in `_NET_WM_NAME`). Consequently, once the `PropertyNotify` event will be received, it will contain the timestamp needed to set the ownership on `_NET_CM_S`n.

Third round-trip

1. Last step to initialise the extensions by calling `display_init_extensions_finalise()` function. It checks for versions of the X extensions on the server-side (at least 0.2 for

the COMPOSITE extension to get `NameWindowPixmap` support and from 2.0 for the XFIXES extension to get support of `Regions`);

2. Load the effect plugins into the program address space by calling `plugin_load()` function to perform a `dlopen()` on plugins set up in the configuration file. If the plugin has been successfully loaded, it looks for the virtual table symbol, namely `plugin_vtable` (see section 4.2 for further details);

3. Send the `GetModifierMapping` request to get the keyboard mapping meaningful to determine the `KeySym` from a `KeyCode` returned by either a `KeyPress` or `KeyRelease` event;

4. Finalise the registration of the compositing manager by calling `display_register_cm_finalise()` function after handling the `PropertyNotify` event in the queue whose handler sends a `SetSelectionOwner` request where the timestamp is given in the event itself.

   ICCCM manual specifies that the program should check whether the ownership has been successfully taken, therefore a `GetSelectionOwner` request is sent by calling `xcb_ewmh_get_wm_cm_owner()` function within the handler. Finally, `display_register_cm_finalise()` gets the reply of this request and checks whether the XID of the reply matches the compositing manager window.

Fourth round-trip

1. Redirect windows to the off-screen buffer by issuing a `RedirectSubwindows` COMPOSITE request and manage existing windows (as described in section 2.3.3) by calling `display_init_redirect()` function after sending a `GrabServer` request to ensure there will be no race condition while performing these tasks. It also sends a `ChangeWindowAttributes` request on the root window in order to declare interests in events described in section 2.3.4;

2. Check requirements of effect plugins and enable them accordingly. This process is described in further details in section 4.2;

3. Process the reply to get the keyboard mapping;

4. Reset the events and errors handlers to the common normal events and error handler by calling `event_init_handlers()` function. The errors are only displayed as warnings which should be investigated anyway (some errors are sometimes hard to figure out due to the asynchronous model of the program) but are not considered as fatal in contrary to the startup error handler. The event handlers are described in section 2.3.4.

2.3.3 Managing windows

All the windows are stored as a linked-list of windows objects (namely `window_t`) which contains the following fields:

- The XID as given when querying the list of windows on initialization or when receiving a `CreateNotify` event. The XID is used in all requests relevant to a window;
- Window attributes as returned by `GetWindowAttributes` request meaningful to get the map state, `Visual`, override redirect status and the window class. `InputOnly` windows are never painted on the screen because they are always invisible when the windows are not redirected to the off-screen storage;
- Window geometry as returned by `GetGeometry` request (or updated by the handler of `ConfigureNotify` to avoid an unnecessary request) used to paint the window and its border at the proper coordinates;
- The `Damage` object associated with the window XID along with a boolean stating whether the window has been damaged (the meaning of this object and field will be explained in section 2.3.4);
- The window `Pixmap` as returned by `NameWindowPixmap` COMPOSITE request or by the effect plugins themselves when relevant (for instance to paint thumbnails of windows for EXPOSÉ plugin);
- Rendering backend specific data whose data type is declared in the rendering backend itself (see section 3.2 for further details).

When new windows objects are created, the first three fields are initialized, either on startup when managing existing windows or when receiving a `CreateNotify` event. The `Pixmap` gets initialized only if the windows is visible or when receiving a `MapNotify` event, otherwise it is useless as the COMPOSITE extension has not created it yet.

As `PropertyNotify` events are not reported when the window is not *mapped* whereas a program may change the event mask while unmapped, therefore it is needed to set the event mask when the window is mapped as implemented in `window_register_notify()` function and performed by sending a `ChangeWindowAttributes` request where `value-mask` and `value-list` respectively equal to `event-mask` and `PropertyChange`.

2.3.4 Main events loop

The main events loop firstly calls `xcb_wait_for_event()` function to block until at least one event is received, then handled by calling the appropriate handlers thanks to `xcb-event` library. This first call avoids spinning like it would do with `xcb_poll_for_event()`. All remaining events in the queue are processed by calling `xcb_event_poll_for_event_loop()` non-blocking function.

The following events are handled by relying on `xcb-event` library:

`DamageNotify`	Send a `DamageSubtract` to set back the damaged region as empty, therefore making DAMAGE extension reports `DamageNotify` the next time the window will get *damaged*. It also sets the window object as *damaged* to paint it on the on-screen buffer.
`KeyPress` `KeyRelease`	Propagate the event to the plugins because the core code does not define any shortcuts, but actually handled by the plugins itself.
`CirculateNotify`	Update the global windows list according to the new stack position of the window.

Continued on next page

`ConfigureNotify`	Update the root window geometry and resets its background in case the event is related to the root window. Otherwise, it updates the window object attributes, geometry and stack position. If the window geometry has changed, the `Pixmap` associated with the window object (usually coming from `NameWindowPixmap` COMPOSITE request) is destroyed and fetched again as it is not consistent with the window content anymore.
`CreateNotify`	Call `window_add()` function and get the window geometry as given in the event itself.
`DestroyNotify`	Free the memory and X data (including rendering backends and effect plugins) associated with the window object.
`MapNotify`	Update window attributes and free the `Pixmap` associated with the window as a new one is created when a `Window` is *mapped*.
`ReparentNotify`	Manage the window if the parent has been changed to the root window or remove it from the windows list in case of it was previously a child of the root window. Indeed, only children of the root windows are considered, otherwise they are ignored because a given window may have several children.
`UnmapNotify`	Update windows attributes and set the window as not *damaged* as only the content of mapped windows is now considered invalid.
`PropertyNotify`	Reset the root background if it is a root background window property. if the `Atom` is `_NET_SUPPORTED`, it updates its value as it is necessary for effect plugins requirements. Then, it also checks whether the plugin requirement can now be met by this event, if so the plugin is enabled (see section 4.2 for further details about plugins requirements).
`MappingNotify`	Update the keyboard mapping stored as a global variable.

Table 2.4: Events handling in core code

If the screen has to be repainted, it iterates through the `render_windows` hooks of effect plugins, if there is one returning a non-`NULL` value, then it is used as the list of windows, otherwise it uses the list of windows stored as a global variable, then paint them on the screen by calling `window_paint_all()` function from `src/window.c`. This design allows a plugin to *take over* the windows of the core code and paint on the screen whatever it wants (for example EXPOSÉ plugin only paints thumbnails of the windows but not the original windows).

Before going to the next iteration of the loop, it calls `xcb_aux_sync()` to ensure all the requests in the queue have been sent and processed to make sure everything has been painted on the screen.

2.4 Issues and shortcomings

`xcompmgr` keeps a *damaged* region which is the union of the regions reported by DAMAGE extension and allows to avoid useless copies of pixels when painting on the screen, and consequently improving the overall compositing performance a lot. However, it has not been written yet because a lot of time has been spent on other core parts of the program but it should be soon as it is quite slow at the moment. Once it will have been written, I will release a first stable version and will announce it on relevant mailing lists.

A window manager may be define in two ways when managing windows. It may either leave the immediate children of the root window as they are or it may re-parent them to another window in order to provide decorations and title bars. The former solution is used by RATPOISON[2], AWESOME and XMONAD[3] because they do not provide any decorations or title bars, whereas the latter is used by most window managers nowadays. At the moment, the support for the latter one is not complete and has not been widely tested yet because I decided to focus on getting it to work properly on the window manager I am using, namely Awesome. However, supporting re-parenting window managers properly will be done after the overall performances and stability will have been improved.

If acquiring ownership on `_NET_WM_CM_S`n root window property fails, the program cur-

[2] `http://www.nongnu.org/ratpoison/`

[3] `http://xmonad.org/`

rently exists but it would be better in future releases to implement a `replace` program parameter to *force* taking ownership on this property.

Nonetheless, it is worth noticing that at the moment the labels of requests and errors are stored statically as an array of strings but this is not convenient at all as each program which would like to display error message as a string instead of a plain code must implement the same code again and again. XLIB provides an `XGetErrorText` function to get the label message from the error codes and handling errors from core and extensions requests, but XCB does not provide such feature. Therefore, it would be interesting to implement a library within `xcb-util` to achieve the same purpose as XLIB function. However, this implementation needs to be discussed before with other XCB maintainers as it would probably mean gathering errors and requests labels from the XCB XML files.

As the list of windows and plugins are implemented as linked-lists, it may be worth improving these algorithms, for instance by using arrays and identify the most common operations for a given data structure (for example the list of plugins is not often expanded but really often browsed). In addition, whenever possible I tried to not use heap allocations but stack allocations because the former is much slower and often leads to memory leaks, but the number of heap allocations may still be reduced.

2.5 Future features

The current architecture may be extended to be more flexible because currently a plugin such as EXPOSÉ may be activated only by setting up the key pressed in the compositing manager configuration file. However, by supporting an interprocess communication system such as D-BUS[4], it may be possible to interact tighter with the window manager for example to allow activation of plugins when clicking on an icon. This way, such message could be sent directly and easily by the window manager or any other program supporting D-BUS or thanks to `dbus-send` command.

[4] `http://www.freedesktop.org/wiki/Software/dbus`

Chapter 3

Rendering backends

This chapter describes rendering backend rationale and architecture which sits on top of the actual rendering engine used. Only a RENDER backend has been implemented so far but it could be extended easily in the future to implement other rendering methods as explained in section 3.5.

3.1 Rationale

Nowadays, most compositing managers only relies on only one method to paint the windows on the screen, usually RENDER (e.g. `xcompmgr`) or OPENGL (e.g. `Compiz`). However, it could be meaningful to provide different rendering backends (thus making the core and plugins code independent on the rendering method used) and let the user chooses according to its requirements and needs.

Firstly, graphic drivers on X WINDOW SYSTEM-based operating systems usually provides different performances according to the rendering method used (commonly RENDER or OPENGL). For example, most graphic card vendors (such as Nvidia) do not provide the full specification of their products, therefore the free drivers acceleration written by the community tends to be slower for OPENGL than proprietary drivers written by graphic card vendors whereas RENDER behaves reasonably well.

Secondly, embedded devices usually do not usually implement RENDER nor OPENGL but rather OPENGL ES designed for this purpose. Furthermore, embedded devices gets more and more powerful and popular as PDA or mobile phones. Therefore, it could be interesting to provide a specific OPENGL ES rendering backend which would meet embedded devices needs.

Thirdly, even if most effects implemented in this compositing manager only rely on 2D graphics, effects implemented in the future may want to take advantage of 3D, therefore ending up relying on OPENGL or OPENGL ES.

3.2 Architecture

The rendering backends are implemented as dynamic libraries loaded on runtime with `dlopen()` and unloaded when exiting the program by calling `dlclose()`. The rendering backend to be loaded can be set up in the configuration file (`rendering` option as explained in further details in section A.6.2).

The data related to the rendering backend is stored directly in the global variable structure (namely `globalconf`) as a `void` pointer (names `rendering`), whose actual type is defined in the backend itself, because obviously only one rendering backend can be used at the same time. In addition, per-window data are stored in the window object (names `rendering`) itself as a `void` pointer too as the data type is defined in the rendering backend itself.

Before getting into details about the rendering backend, it is worth mentioning that the background image is stored as a `Pixmap`. It is usually set up by a third-party program (such as `wmsetbg` from WINDOW MAKER[1] or `Esetroot` from ETERM[2]) or by the window manager itself. By convention, the `Pixmap` XID is commonly stored in both `_XROOTPMAP_ID` (most common) or `_XSETROOT_ID` root window property.

The rendering backend draws to a buffer (called the *root buffer*) and to the root window at the end on purpose to avoid flickering if the window would have been painted directly on the root window. A rendering backend exports at least a `rendering_t` structure, whose symbol

[1] `http://windowmaker.info`

[2] `http://www.eterm.org`

name is `rendering_functions` and holds the following hooks implemented as pointers on functions:

- Initialization routine called on startup and split in two routines, namely `init` and `init_finalise`, which allows to send requests and get their replies asynchronously. These routines initialise the X extension and also get the root background `Pixmap`;
- `paint_background` paints the root window background on the root buffer;
- `paint_window` paints a window object and more precisely the window object `Pixmap` on the root buffer. This routine is also responsible of painting the alpha channel of the window where its value is returned by the `window_get_opacity` plugin hook, explained in further details in section 4.2. Window objects data associated with windows and related to the rendering backend are initialized in this routine;
- `paint_all` paints the content of the root buffer to the root window;
- `reset_background` is called when a `ConfigureNotify` event related to the root window has been received or when the background `Pixmap` has been updated (notified by a root window `PropertyNotify` event of ones of the background atoms listed above);
- The rendering method used defines its own X request and errors, therefore it is needed to provide functions to nicely displayed which request failed and what was the error message. `is_request`, `get_request_label` and `get_error_label` hooks aim respectively to determine whether this is an error related to the rendering backend by looking at the extension major code of the request, get the label associated with a request from its extension minor code and the error label given by the error code;
- `free_window_pixmap` frees the data structures allocated for a window `Pixmap` stored in the window object;
- `free_window` frees the `rendering` pointer stored in the window object which cannot be accomplished by the core code itself as it is not aware of the type used.

Once the rendering backend is loaded by a `dlopen()`, the address of the `rendering_t` structure is obtained by calling `dlsym()` function stored in the core code global structure.

The hooks defined at the moment are sufficient enough for RENDER implementation described in the next section but it might evolved in the future to suit other rendering methods needs such as OPENGL and OPENGL ES (further details are given in section 3.5).

3.3 X RENDER extension

3.3.1 Overview

This extension brings server-side digital image composition of geometric figures (Packard 2005-07-01, Packard 2000-04-19, Höglund n.d.). Unlike the core X WINDOW PROTOCOL, `Render` extension has an understanding of alpha channel and basically provides only the following operation to generate an images (`dest`) from `source` and `mask` images (where `IN` is the Porter and Duff operator (as explained in section A.1) and `OP` is an operator listed in `Render` specification which includes Porter and Duff operators):

$$dest = (source\ IN\ mask)\ OP\ dest \tag{3.1}$$

Indeed, the rendering system implemented in the core X WINDOW PROTOCOL is pixels aware only whereas a compositing model operates on colors in `ARGB` format. Therefore, `Render` defines a new object, `PictFormat`, to translate pixel values into `ARGB` values. This object is associated with a `Drawable` into a new `Picture` object.

From a compositing manager point of view, `Render` provides requests to access window contents needed for alpha blending. It was primarily used before to implement anti-aliased fonts but heavily used nowadays to implement visual effects such as drop shadows and translucency for example.

3.3.2 Operators and requests

Among the operators defined by `Render` (`OP`) for `Picture`, only the following ones really matter for a compositing manager:

- `PictOpOver` corresponds to the Porter and Duff `Over` operator and may be used to achieve windows translucency where `dest` window covers `src` window. When applying this operator, the expression (3.1) becomes $dest = src\ Over\ dest$;
- `PictOpSrc` specifies that `dest` should be replaced by `src` including alpha channels, thus it simply applies opacity on `dest` window. For example, this operator may be used to achieve windows fading in a compositing manager. When applying this operator, the expression (3.1) would result in $dst = src$ where src is dst with an alpha channel.

Among the requests provided by this extension, the following ones are relevant when writing a compositing manager relying on `Render` and are used in `xcompmgr`:

CreatePicture(Drawable drawable, PictureFormat format, BITMASK value-mask, LISTofVALUE value-list) => PICTURE pid	
Explanation	Create a `Picture` `pid` object associating `format` with `drawable`. `format` specifies how the pixel should be converted to a RGB value according to the `Visual` as defined in 1.1.8 section. `value-mask` and `value-list` specify the new `pid` attributes. Among the possible values for `value-mask` and `value-list`, there are `IncludeInferiors` (`subwindow-mode` mask) to include the child widgets in the window when drawing it and `repeat` (`Repeat` mask) to indicate how `drawable` contents is extended in both directions. Once a `Picture` is not used anymore, it has to be freed on the server-side by issuing a `FreePicture` request.
Compositing Manager	Used each time for `Drawables` (either a `Window` or a `Pixmap`), such as windows and root background, before painting them on the screen by issuing a `Composite` RENDER request which requires `Picture` objects.

Continued on next page

Composite(PictOp op, Picture src, Picture mask, Picture dst, INT16 dst-x, INT16 dst-y, CARD16 width, CARD16 height) [3]	
Explanation	*Combine the specified rectangle of the transformed* `src` *and* `mask` *operands with the specified rectangle of* `dst` *using* `op` *as the compositing operator* (Packard 2005-07-01).
Compositing Manager	Allow to paint the given `Pictures` on the screen by specifying usually `PictOpSrc` operator for opaque windows or to paint the root background for instance. Translucency visual effect may be achieved by specifying a `mask Picture` including only the alpha channel and `PictOpOver` operator.

Table 3.1: RENDER X extension requests

3.3.3 Hardware acceleration

`Render` may be hardware-accelerated thanks to either XAA (Hanemaayer 1996) (XFREE86 ACCELERATION ARCHITECTURE), EXA, UXA or GLUCOSE graphic acceleration architecture (Smirl 2005-08-30). Given that desktop usage has evolved a lot since the beginning of the X WINDOW SYSTEM project, graphic acceleration architectures have evolved accordingly to suit users needs. At the beginning, there was XAA which has been deprecated by EXA and in turn by UXA and GLUCOSE for the reasons stated in the next sections.

These architectures both aim to accelerate most common drawing operations by providing a *driver interface that allows the driver to communicate its acceleration capabilities and restrictions back to the graphic acceleration architecture* (Eich 2004-04-23). Complicated drawing operations (also referred as X primitives) which are too complex to be supported can be broken in simpler primitives. When rendering tasks cannot be accelerated, it simply fallbacks on software rendering.

[3] This request allows more parameters but only the relevant ones are given.

3.3.3.1 XAA

XAA was the first acceleration graphic architecture implemented in the X WINDOW SYSTEM and implements 2D hardware acceleration in user-space for portability reasons years before RENDER X extension. Each graphic driver has to be updated to support XAA which represented a huge amount of work.

As stated in (Jackson 2005-04-12), XAA is not suitable for modern desktop usage for the following reasons:

- As it is implemented in user space, the kernel is unaware of X operations on the graphic card, thus leading in conflicts with the console driver and also a performance bottleneck on context switching between these drivers;
- It accelerates rendering operations rarely used nowadays;
- Support of RENDER X extension is really poor and slow as showed clearly by `xcompmgr`. Firstly, the off-screen `Pixmaps` have to be exactly of the same format of the displayed screen. Secondly, it cannot work on `Pictures` both in graphic card memory as the source has to be in host memory whereas the destination has to be in the graphic card memory.

3.3.3.2 EXA

EXA is the XAA successor designed to address the last two issues mentioned in the previous section by providing a more advanced graphic driver and memory management API. Therefore, it provides a better integration with `Render`, for modern desktops with a compositing manager, without needing major changes in existing graphic drivers already implementing XAA. Nowadays, EXA is available for most common graphic cards (Jackson 2005-08-21).

Although the performances of EXA were not really good when it was started in 2005, it has evolved lately and now behaves really well with major graphic drivers according to benchmarks (Worth 2009-02-19). Moreover, it is still being improved from a performance point of view.

3.3.3.3 UXA

UXA is the future of X hardware acceleration based on GEM[4], which addresses all the issues of XAA mentioned in section 3.3.3.1. GEM is a modern memory manager especially designed for graphic device drivers and implemented in the LINUX kernel which aims to address the first issue of XAA by allowing resources to be shared. UXA is originally based on EXA without the useless code already implemented in GEM.

There is still a lot of work to be done before being available for all graphic cards and especially performances improvements but the project is pretty recent. In fact, the development is focused on getting INTEL graphic cards working at the moment, however RADEON and VIA S3 graphic cards support has been merged recently in LINUX kernel, thus making UXA a promising acceleration architecture once performances will have been improved.

3.3.3.4 GLUCOSE

GLUCOSE differs from others graphic acceleration architectures as it intends to accelerate X rendering operations thanks to OPENGL. It is based on XGL code which is an X SERVER on top of OPENGL and GLITZ (an efficient RENDER-like extension on top of OPENGL (Peter Nilsson June 27July 2, 2004)), finally deprecated in favor of AIGLX (ACCELERATED INDIRECT GLX). Existing graphic drivers require to be updated like other graphic acceleration architectures, which can be achieved by simply initializing GLUCOSE.

At the moment, GLUCOSE is focused on INTEL graphic cards only and performances are not really good at the moment (Worth 2009-06-10) but it is quite normal as it is currently a work in progress.

Moreover, GLITZ provides an EGL backend which is an **architecture-independent** interface between a rendering API such as OPENGL or OPENGL ES and the underlying native platform window system such as X WINDOW SYSTEM (Group 2009).

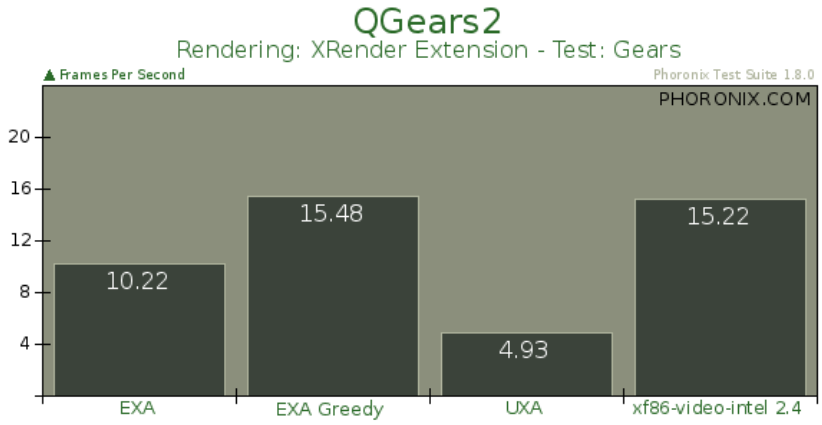

Figure 3.1: *Rendering benchmark (Gears) with* RENDER *extension for* EXA *and* UXA

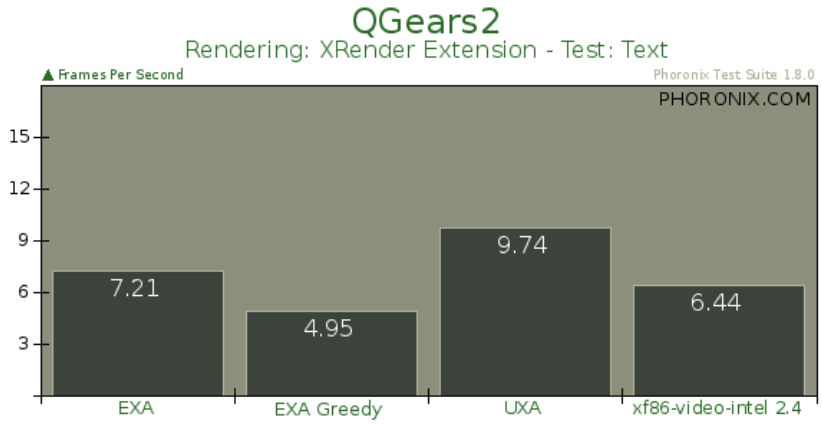

Figure 3.2: *Rendering benchmark (Text) with* RENDER *extension for* EXA *and* UXA

3.3.3.5 Performance comparison

This section aims to compare performances of existing graphic acceleration architectures, namely EXA and UXA (but not GLUCOSE as this is highly experimental and some benchmarks already exist (Worth 2009-06-10) and not XAA as it is now obsolete) on INTEL graphic cards which are the only one supporting without important bugs. These results are provided by PHRONIX.COM [5] and compare UXA with different versions of EXA (the most recents ones are

[4] `http://keithp.com/blogs/UMA_Acceleration_Architecture/`

[5] `http://www.phoronix.com/scan.php?page=article&item=ubuntu_intel_greedy&num=1`

EXA and EXA GREEDY whereas XF86-VIDEO-INTEL 2.4 is older). I was not able to run my own benchmarks because I only have a ATI RADEON graphic card and the UXA support is highly experimental (actually I was not able to make it work properly).

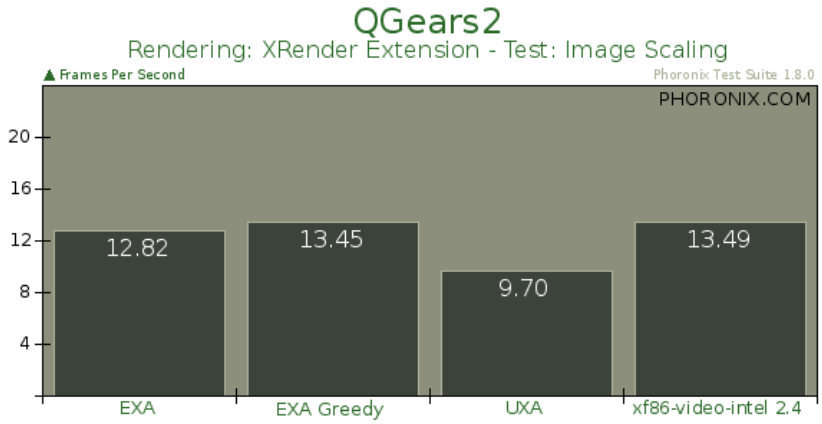

Figure 3.3: *Image scaling benchmark with* RENDER *extension for* EXA *and* UXA

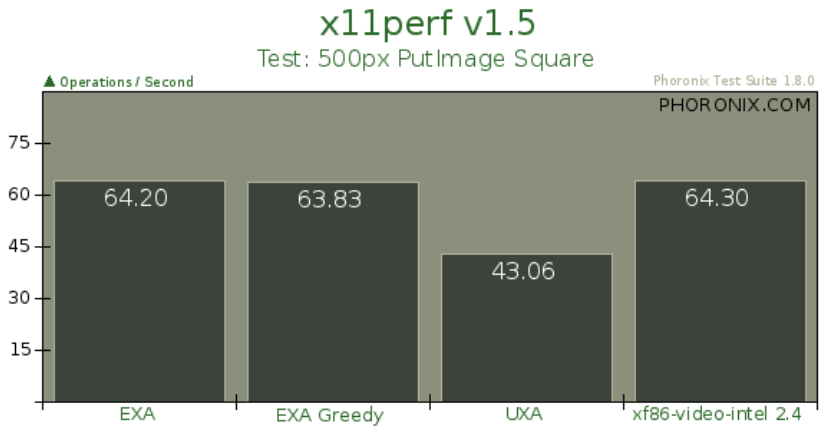

Figure 3.4: *PutImage benchmark for* EXA *and* UXA

UXA behaves reasonably badly with Gears (as showed on figure 3.1) and image scaling (as showed on figure 3.3) and with PUTIMAGE request (as showed on figure 3.4 and meaningful for effect plugins such as EXPOSÉ) whereas it behaves well for text as showed on figure 3.2. Therefore, we can consider that UXA behaves badly at the moment, but these benchmarks have been performed some months ago and a lot of work has been achieved in the meantime. However, UBUNTU decided to not ship UXA in their latest release because of these performances issues.

3.4 RENDER backend

This section describes implementation of a rendering backend based on RENDER extension available in `rendering/render.c` source file. Besides the X render extension, this backend also rely on `xcb-util/renderutil` (see section A.2.1.7 for further details).

3.4.1 Data structures

The global rendering structures (`_render_conf_t _render_conf` structure) holds the following data to cache the results of requests which are not likely to be invalidated often:

- Reply of `QueryVersion` asynchronous request which includes the extension major opcode, the index of the first event and error. All these information are relevant for error reporting and especially because according to the RENDER specification it is compulsory to issue this request, otherwise the behavior of the server is undefined;
- The RENDER `Picture` objects associated with the root window, the root buffer `Pixmap`, and the root background `Pixmap`. These pictures are not modified often so it is worth storing them globally;
- `Picture` formats and `Visual` supported by the screen as these values will not change once initialized.

The per-window rendering data only stores the `Picture` associated with the window `Pixmap` and alpha `Pixmap` if relevant.

3.4.2 Hooks

This section describes the different hooks which may be called from the core code.

3.4.2.1 Initialization hooks

When the backend is loaded in memory after calling `dlopen()`, the backend prefetches the extension data (`render_preinit` function) used later to check whether the extension is

present on the server-side.

render_init sends a QueryVersion request necessary before sending any RENDER request, QueryPictFormats to get the list of supported PictFormats and associations between PictFormat and Visual. It also sends the request to get the root background Pixmap whose XID is given in Atom XIDs already fetched in the core code (see section 2.3.2 for further details about the root background).

render_init_finalise processes the replies to the requests send in the function described before. It also creates the Picture associated with the root window XID and the root background Pixmap, then it creates the root buffer Picture. Finally this function paints the root background to the root Picture.

All the Pictures created in these hooks and the QueryPictFormats reply are freed in render_free hook.

3.4.2.2 Paint hooks

render_paint_background only sends a COMPOSITE RENDER request from the root background Picture to root buffer Picture specifying PictOpSrc as operator. The background is painted on the screen when calling render_paint_all.

render_reset_background destroys the root background Picture and creates a new one with the updated root background Pixmap which is painted on the root buffer Picture by sending a COMPOSITE request equivalent to render_paint_background.

The first time render_paint_window on a window object, it performs the following steps:

1. Allocate the memory specific to the rendering backend, namely the Picture and alpha Picture associated with the window Pixmap;

2. Create the Picture associated with the window Pixmap which has generally been previously retrieved by sending a NameWindowPixmap COMPOSITE request (this Pixmap may also come from a plugin like EXPOSÉ plugin as explained in 4.4.2 section). The Picture format matches the Visual of the window and the value is initialized to

`subwindow-mode` value set to `IncludeInferiors` to specify that only pixels of the siblings windows will be painted when the window is used as a destination;

3. If the window opacity is not opaque, then a `Picture` for the alpha channel is created. It is associated with a new `Pixmap` and is filled considering only the alpha channel thanks to `FillRectangles` request.

These `Pictures` are discarded when the window `Pixmap` changes which may arise when either a `MapNotify` (the window content is not guaranteed to be preserved while the window is unmapped) or `ConfigureNotify` (when the window geometry changes) event is received.

Finally, `render_paint_window` paints the window `Picture` on the root buffer `Picture`, according to the window geometry, using either `PictOpSrc` or `PictOpOver` depending on the window opacity.

`render_paint_all` is responsible of drawing the root buffer `Picture` into the root window `Picture` by sending a COMPOSITE request on the root window area with `PictOpSrc` as operator.

3.4.2.3 Errors handling hooks

There is nothing really specific to the RENDER backend as this is common to any X extension. Like `src/event.c`, the labels of requests and errors are stored statically as an array of strings (see section 2.4 for further details about this issue).

3.5 Future backends

3.5.1 Overview

In the future, I am planning to write additional backends based on OPENGL (based on `GLX_EXT_texture_from_pixmap` operation of GLX (Leech n.d.) extension specific to the X WINDOW SYSTEM to get an OPENGL `texture` from an X `Pixmap`) for the reasons mentioned in section 3.1. Most of the time, UNIX operating systems use MESA implementation

of OPENGL, therefore this backend would use this library. This could be achieved by linking the new OPENGL backend to an X11/XCB-capable MESA library which is generally the case nowadays, this will make X11 using XCB as a backend, as it could not use XLIB because the libraries does not share the same event queue, and especially because it would be tedious to use GLX extension directly concerning DIRECT RENDERING (Salminen 2009-06-26). Then, it is just about calling OPENGL functions defined in MESA headers.

3.5.2 OPENGL libraries

Another approach than using MESA would be to rely upon a third-party library such as GLITZ, CLUTTER or EVAS which aim to provide a library on top of MESA (these libraries are briefly described in section A.3).

All these libraries provide good performances and are implemented using XLIB. However, GLITZ is certainly the lowest-level one because it is basically about implementing the same RENDER operations, whereas EVAS and CLUTTER are easier to use.

GLITZ is written using XLIB but may be used through CAIRO (however the GLITZ back-end for the latest version of CAIRO is currently a work in progress). As CAIRO provides an XCB back-end too, it could have been possible to use GLITZ through CAIRO, but unfortunately, `cairo-xcb` is outdated and would need to be merged with `cairo-xlib` which would represent an huge amount of work.

Some of these libraries implement OPENGL ES (for example CLUTTER) and even EGL (for example GLITZ supported on compilation) and visual effects out of the box, so it may be worth using such libraries to minimize the code to write and provide good performance as well.

Chapter 4

Effect plugins

This section describes plugins architecture and plugins which have been implemented, namely OPACITY and EXPOSÉ.

4.1 Rationale

`Compiz` (Wikipedia 2009-03-09*a*) implements a plugin architecture for effects too and it has been a good and productive idea which allowed a lot of people to contribute to the project without necessarily understanding the (pretty huge) core code.

A plugin architecture for visual effects provides the following advantages compared to a monolithic approach:

- Avoid modifying the core code which may introduce bugs;
- Many plugins may be implemented without increasing the main binary size;
- Allow to easily load or unload plugins on-demand;
- Allow to implement new effects by just having a basic knowledge of the core code.

The only drawback of such architecture is common to any modular architecture, namely it makes the interactions between the core code and the plugins more complex but it is worth doing

it anyway considering the pros listed above.

4.2 Architecture

The plugins are actually dynamic libraries which are dynamically loaded in memory on runtime with `dlopen()` and unloaded by calling `dlclose()`. The plugin may be enabled, set up and disable easily through the configuration file thanks to `plugins` configuration file option and sections relative to the plugins (the syntax and an example of such configuration file is given in section A.6.2). The core code for plugins is available in `src/plugin.c` source code file.

Contrary to rendering backends, all the variables associated with variables from the core code (such as windows) are managed by the plugin itself to be more flexible as plugins may want to maintain complex data structures.

All plugins will rely on EWMH atoms (Group 2006-09-29) implemented by the window manager itself. The atoms supported by the window manager are defined in **`_NET_SUPPORTED`** atom initialized as a root window property (as described in the main event loop of the core code in section 2.3.4). The atoms values are updated in each plugin code at the moment (a better solution which will be implemented in the future is given in section 4.5).

The list of plugins is stored as a linked-list (`plugin_t` which is a structure data type) containing for each plugin the opaque handle for the dynamic library returned by `dlopen()`, a boolean stating whether the plugin is enabled and the plugin virtual table as described below.

Once the plugin is loaded by a `dlopen()`, the address of the virtual table in memory is obtained by calling `dlsym()` function on a specific symbol defined by each plugin, namely `plugin_vtable` of type `plugin_vtable_t`. This virtual table contains the plugin name, event hooks called after the core code events code and other general functions (implemented in C programming language with pointers on functions) as listed in table 4.1.

Hook name	Comments
`check_requirements`	Call before enabling the plugin to allow it to specify conditions before activation (for instance, it allows to check whether its required atoms are actually supported by the window manager, otherwise the plugin would be disabled until one or several `PropertyNotify` events assert the conditions required to enable it). If the plugin meets its requirement, it is set as enabled in its `plugin_t` structure.
`window_manage_existing`	Call on window manager startup to allow the plugin to manage existing windows and initialize their per-window specific data.
`window_get_opacity`	Allow plugins to define the opacity of a given window object. This hook is currently used for `opacity` plugin to returns the opacity associated with a window.
`render_windows`	Allow plugins to specify its own windows which are going to be rendered by the rendering backend. For example, a plugin may perform transformations on the windows and then paint them on the screen instead of the actual windows like EXPOSÉ plugin does.

Table 4.1: Effects plugins hooks

A plugin may also defines constructor and destructor respectively called on `dlopen()` and `dlclose()`. However, the constructor of a plugin cannot assume that the windows list or the rendering backend have been initialized, therefore the constructor must only allocate memory or send X requests independently of existing windows or rendering backend.

4.3 OPACITY plugin

4.3.1 Overview

Before compositing managers, on X WINDOW SYSTEM, window transparency had to be implemented by the client itself and was called *false transparency*. Basically, it was about copying the background pixmap from the root window to the client window, thus involving several requests between the server and the client. In addition, it was slow because this operation was performed directly on the display using the CPU instead of taking advantage of the GPU.

This feature is really common nowadays and available on almost all operating systems because windows opacity relies on a basic mechanism, namely compositing, used by all compositing managers. The figure 4.1 shows an example as implemented in `xcompmgr`.

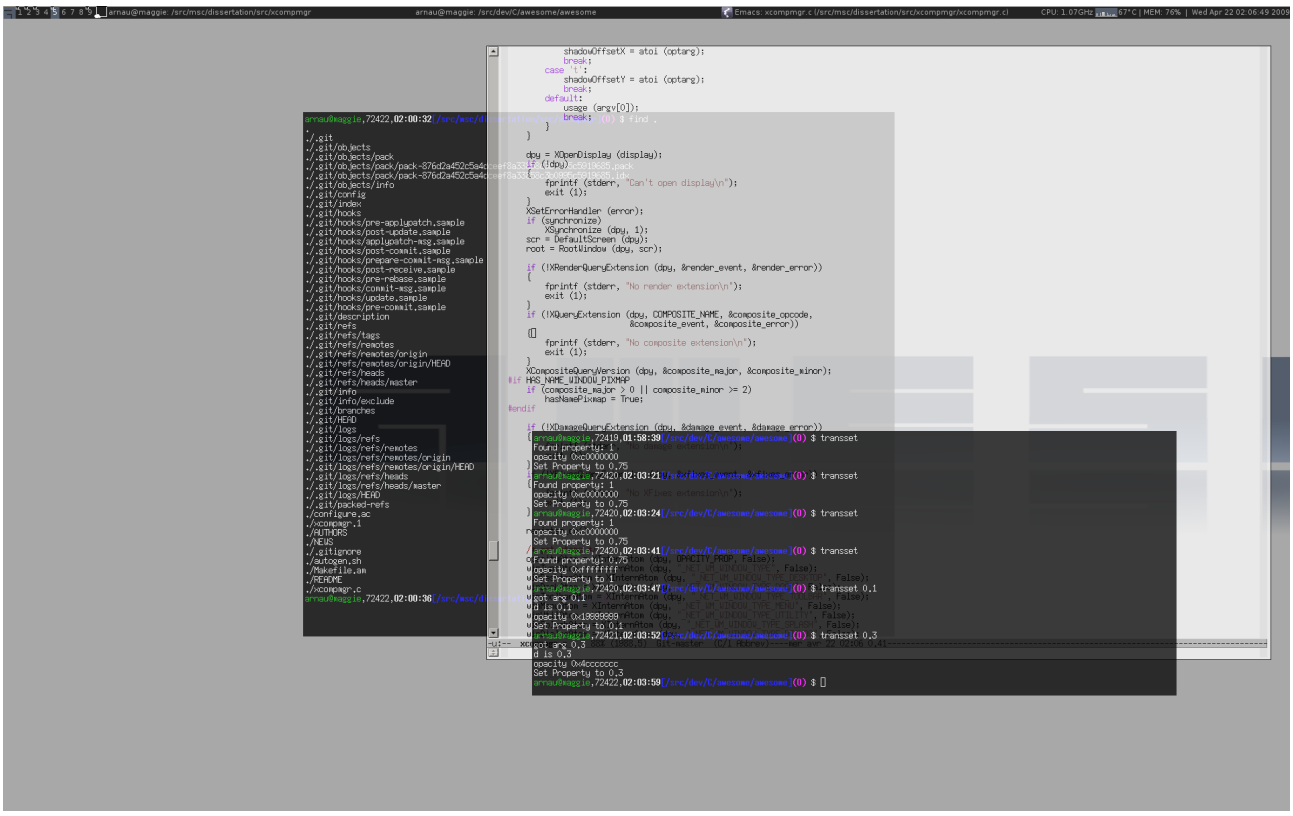

Figure 4.1: *True transparency with* `xcompmgr`

4.3.2 Implementation

This plugin relies on **_NET_WM_WINDOW_OPACITY** atom set as a window property. This property is usually set in the window manager configuration or by specifying the opacity to the `transset` command and then pointing the window with the mouse. If this property is not set on a window, then the window is considered opaque by default (there is no need to provide a configuration option for opacity as it is usually the window manager responsibility). The source is available in `plugins/opacity.c` file.

This plugin stores the opacity property which is associated with a `window_t` object along with the `GetProperty` request cookie. Only the *mapped* windows opacity properties are stored because the value of a window property may changed while the window is *unmapped*, however the server would not send a `PropertyNotify` event on an *unmapped* window.

On program startup for all windows or when the plugin receives a `MapNotify` or a `PropertyNotify` event for a specific window, it gets the opacity property in a lazy way because only the `GetProperty` request is sent at this stage. The reply is actually retrieved when the rendering backend asks for it, therefore it allows to take full advantage of XCB asynchronous model.

When the plugin receives a `UnmapNotify` event, it only frees the reply if a request has been previously sent and the plugin data associated with the window being *unmapped*.

The figure 4.2 shows the compositing manager running with opacity sets to values ranging from 0.3 to 0.75 thanks to `transset` command line tool.

4.4 Exposé plugin

4.4.1 Overview

EXPOSÉ (Wikipedia 2009-03-09*b*) was firstly introduced in the MAC OS X operating system (and more precisely in QUARTZ COMPOSITOR (Wikipedia 2009-01-28)) and provides a way to organize and access any windows quickly and easily. Among features included in EXPOSÉ, the most popular one is certainly *All windows* which shows live thumbnails representation of all opened and unhidden windows once a key or button has been pressed and allows to easily

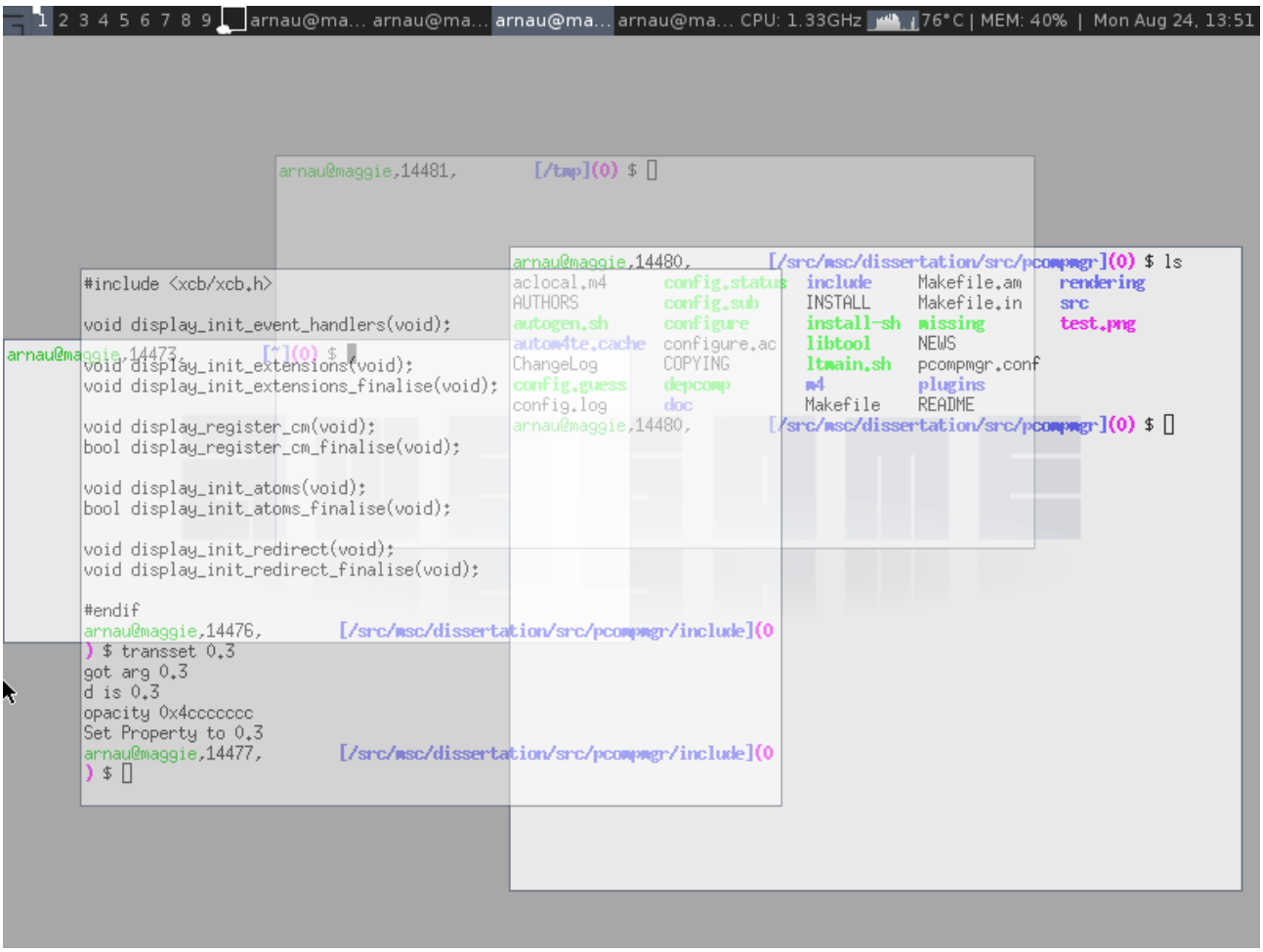

Figure 4.2: *The project compositing manager running with* AWESOME *with* OPACITY *plugin enabled*

and quickly locate and jump to a particular window thanks to keyboard and mouse shortcuts.

This feature has become so popular nowadays that similar features have been implemented in major operating systems such as Microsoft Windows Vista (known as Windows Flip 3D (Wikipedia 2009-02-19)) and X WINDOW SYSTEM-based systems (such as `Scale` plugin for `Compiz` and `Present` plugin for `Kwin` compositing window managers). The figure 4.3 shows an example as implemented in `Compiz Fusion Scale` [1] plugin.

[1] `http://wiki.compiz-fusion.org/Plugins/Scale`

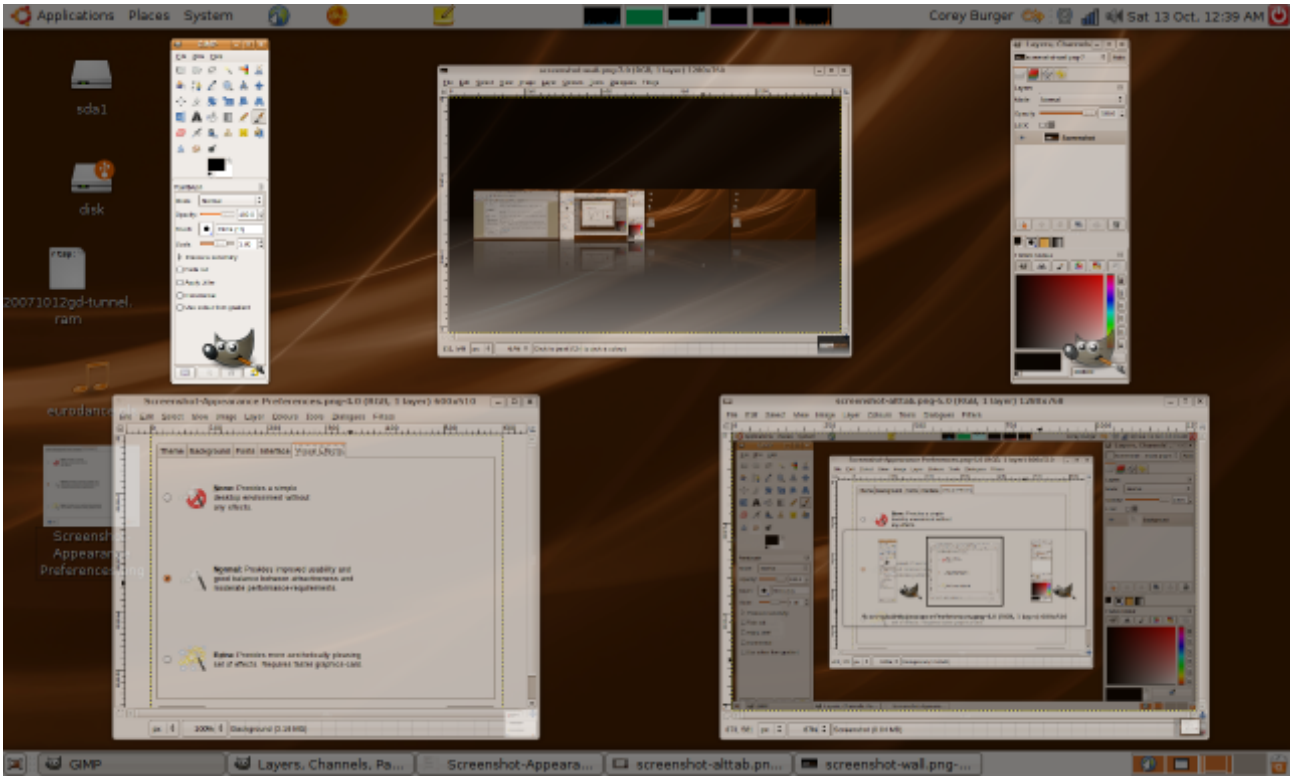

Figure 4.3: *Exposé-like feature in* `Compiz Fusion`

4.4.2 Implementation

The source code of this plugin is available in `plugins/expose.c` file. The figure 4.4 shows the compositing manager running along AWESOME window manager before the plugin gets activated, whereas the figure 4.5 shows when the plugin is actually activated by pressing a key on the keyboard.

4.4.2.1 Requirements

This plugin relies on **`_NET_CLIENT_LIST`** and **`_NET_ACTIVE_WINDOW`** atoms, which are set as properties of the root window, to respectively get the XID of the windows managed by the window manager and the current focused window. Once a window has been selected by either a keyboard or mouse shortcut, it sends a `_NET_ACTIVE_WINDOW` client message to the root window as specified in the EWMH specification, which will in turn be handled by the

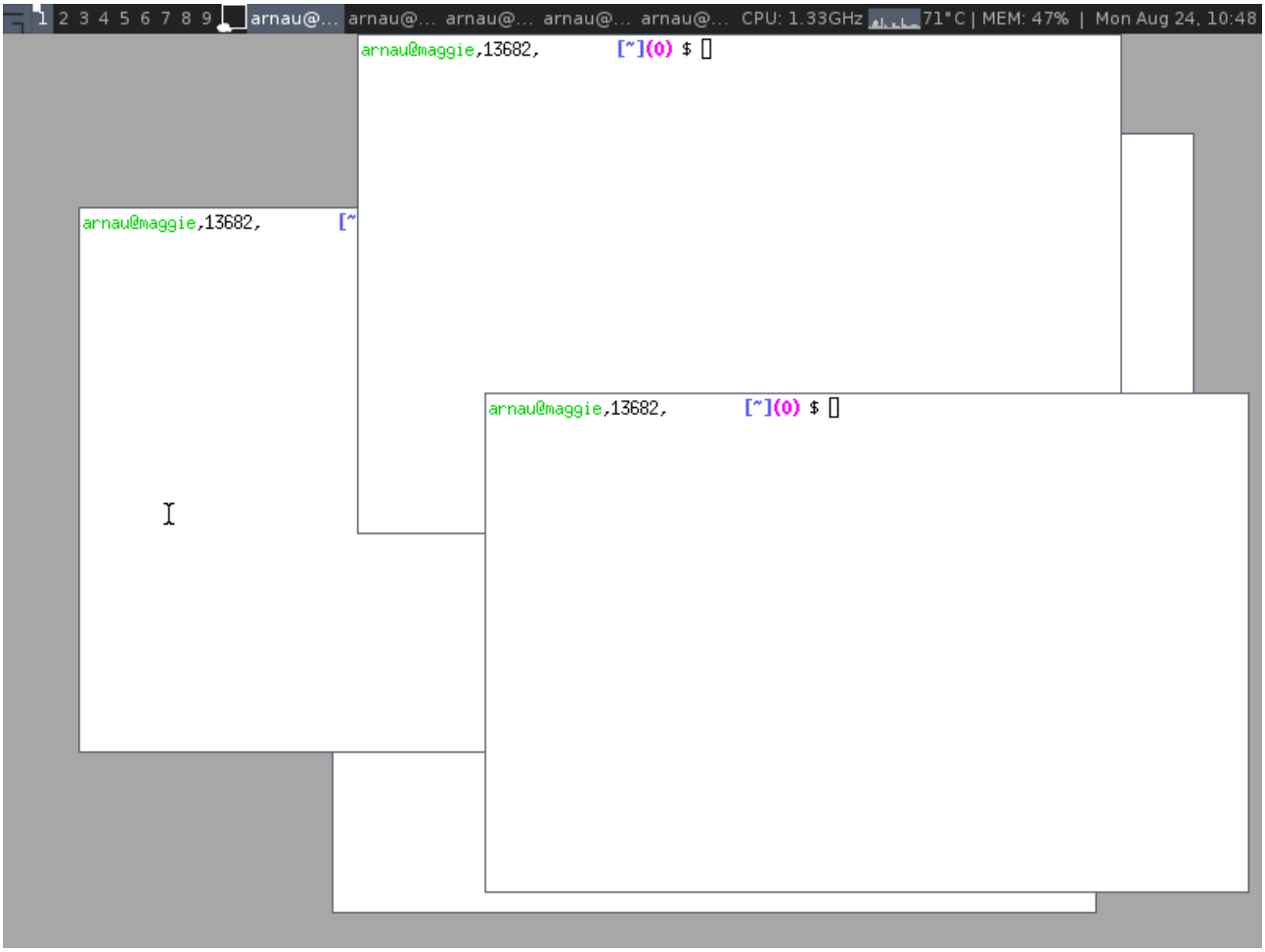

Figure 4.4: *The project compositing manager running with* AWESOME *before* EXPOSÉ *gets activated*

window manager itself to actually switch to the selected window.

Both atoms are required, otherwise the plugin is not enabled, and their values are updated by the core code which also take care of checking whether these atoms are actually supported by the window manager thanks to **_NET_SUPPORTED** root window property.

4.4.2.2 Initialization

When the plugin is enabled by a keyboard shortcut as given in `key` configuration file option (see section A.6.2 for an example) and grabbed passively by sending a `GrabKey` request, the

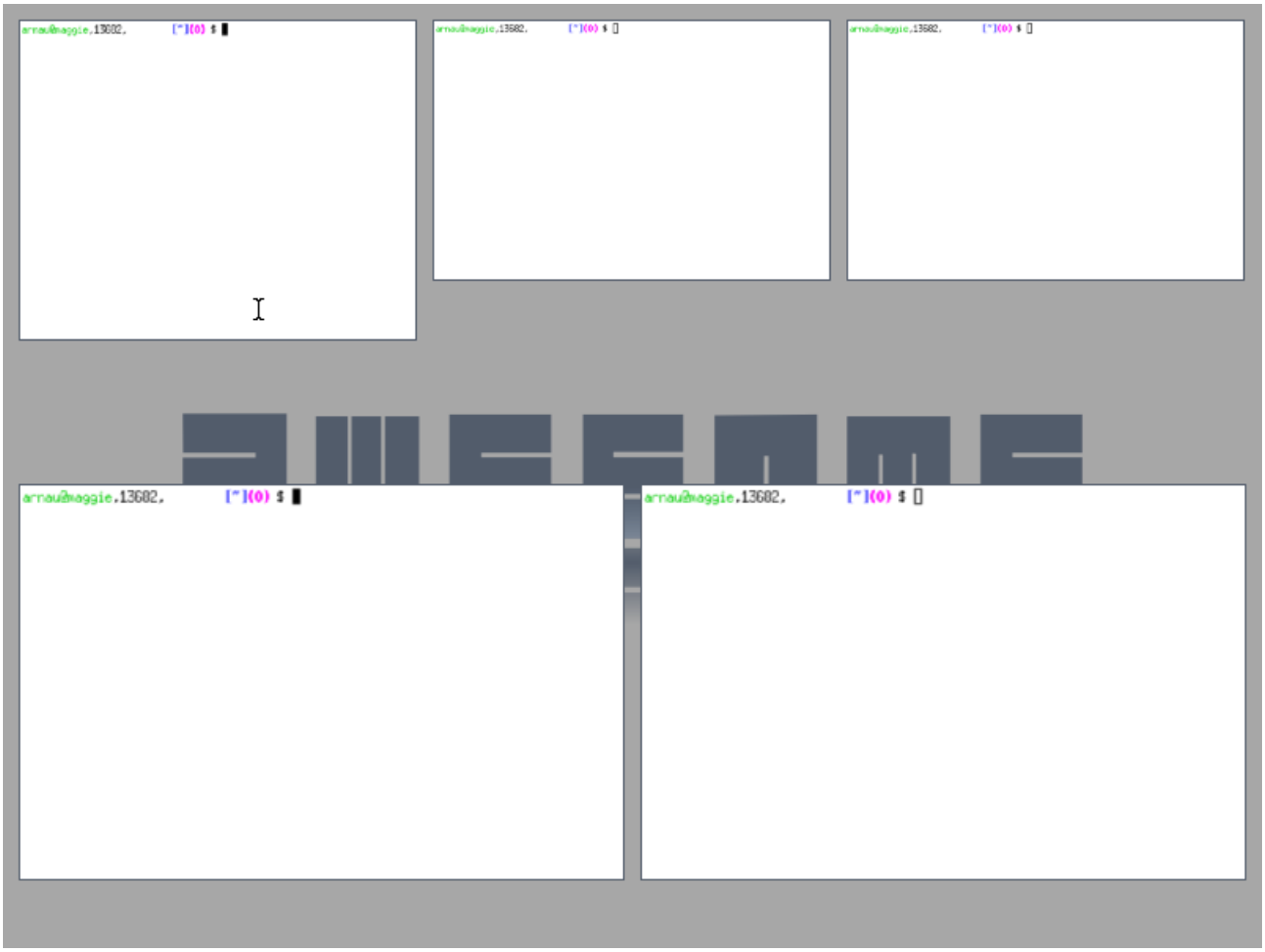

Figure 4.5: *The project compositing manager running with* AWESOME *when* EXPOSÉ *is activated*

following steps are performed:

1. Divide the screen in strips containing slots where the windows managed by the window manager will be put as live thumbnails. The slots are separated from each other by a given number of pixels (`spacing` configuration file option). The number of strips ($strips\_nb$) and average number of slots per strip ($nwindows\_per\_strip$) are computed from the following formulas (where $nwindows$ is the number of windows):

$$\begin{aligned} strip\_nb &= \sqrt{nwindows + 1} \\ nwindows\_per\_strip &= ceil\left(\frac{nwindows}{strips\_nb}\right) \end{aligned}$$

2. Assign each window to a slot based on the Euclidean distance ($distance$) between their respective symmetry center as given by the following formula (where $window\_x$ and $window\_y$ are the coordinates of the window and $slot\_x$ and $slot\_y$ the coordinates of the slot):

$$distance \quad = \quad \sqrt{(window\_x - slot\_x)^2 + (window\_y - slot\_y)^2}$$

3. For each window, it performs the following steps:

   (a) If the window is not already *mapped* on the screen it has to be *mapped* on the screen by sending `MapWindow` requests because the content may be not preserved while it is unmapped on purpose to save memory, thus clients normally never draw on an unmapped window and no `DamageNotify` event would be generated. Therefore *mapping* the window ensures that the window content is up-to-date and that the thumbnails will be updated in real time.

   Before sending this request, it is necessary to send a `ChangeWindowAttributes` request to set `override-redirect` to `true` to avoid the window manager to manage these windows.

   When the plugin is disabled, the window attributes and *mapping* state are set back to their original values;

   (b) Compute the window width and height including the border because `GetGeometry` reply returns the dimension excluding borders;

   (c) Check whether the window needs to be downscaled by comparing the window extents with its slot. If the window has to be rescaled, then create an `Image` and a `Pixmap` of the thumbnail size (computed from the slots size and the original window size by keeping the ratio between width and height), and also a `GraphicContext` associated with the `Pixmap` (see section below for explanation). Finally paint the thumbnail on the screen as described in the next section.

4.4.2.3 Update the thumbnails of the windows

The thumbnails are stored as `Pixmap` painted on the screen by the rendering backend. Before doing so, the original window `Pixmap` has to be downscaled by copying pixels from the

original window. `xcb-util/image` (section A.2.1.5) provides an `Image` object (`xcb_image_t`) and functions to access pixels individually (namely `xcb_image_get_pixel()` and `xcb_image_put_pixel()`) and to put and get an image to and from a `Pixmap` (namely `xcb_image_put()` and `xcb_image_get()`). Therefore, updating the thumbnails will be performed in the following steps:

1. Get the `Image` object from the original window `Pixmap`;
2. Put the pixels on the downscaled `Image`, created during initialization step, from the original window `Image` using a downscaling algorithm described below;
3. Put the downscaled `Image` (excluding the border) on its `Pixmap` using the `GraphicContext`, both created during initialization steps as this size will not changed while the plugin is enabled. Then put the border pixels, if any, on the downscaled `Image`;
4. Set the window as *damaged* to force redrawing of the window by the rendering backend.

The thumbnail `Pixmap` and information such as geometry and original window attributes will be stored in a new `window_t` object. Then, it is given to the rendering backend, therefore it basically replaces the original list of windows by thumbnails of the windows without changing anything in the core code.

4.4.2.4 Algorithm for downscaling an image:

The algorithm written computes a pixel value based on neighboring pixels multiplied by weights according to their position. This algorithm is a simplified version of the Gaussian blur (Wikipedia 2009-06-07) algorithm because it has to be efficient and fast at the same time as there could be a lot of windows to be downscaled. The following matrix is used:

$$\begin{pmatrix} 1 & 4 & 1 \\ 4 & 10 & 4 \\ 1 & 4 & 1 \end{pmatrix}$$

The pixel at the center of the matrix receives the heaviest weight whereas neighboring pixels receive smaller weights, therefore reducing the blurring effect.

Perhaps this algorithm should be included in the core code because a lot of plugins, such as *desktop switcher* and *application switcher*, may want to use it.

4.4.3 Performance improvements

This plugin relies heavily on `Image` objects, however `PutImage` request is a major performance bottleneck as the data have to be copied to the server from the client. Therefore, it would be worth using SHM (SHARED MEMORY) X extension whose usage is described by Corbet (1991). I compared performance of `PutImage` request with and without SHM `x11perf` (the results are available in section A.5) and in most cases, SHM dramatically improves performance by about 45%.

4.5 Issues and future work

At the moment, the plugins to be loaded can only be specified through the configuration file. However, in order to take full advantage of the plugin architecture, it may be worth implementing `enable-plugin` and `disable-plugin` command line arguments to allow the user to either load or unload plugins on runtime. Such feature may be implemented easily when D-BUS support will be implemented with changes mentioned in section 2.5 as it is only about sending a specific message to the running program.

As explained in section 4.2, the EWMH atoms XIDs are stored in EWMH XCB-UTIL library initialized in the core code, whereas the values of the windows properties are managed by the plugins themselves which is not really a problem at the moment as there are only two distinct plugins. However, to avoid duplicated code and requests between future plugins, it would be worth implementing it in the core code. It could be achieved by a dedicated structure holding the atom XID, a `void` pointer (as there are different types of atoms) and a pointer to a function responsible to update the `void` pointer.

The EWMH specification is not really clear about `_NET_ACTIVE_WINDOW` client message behavior because it does not state what to do in case the window to switch to is not on the

current virtual desktop (such confusion has already been reported by GNOME developers [2]). The most common behavior seems to switch to the virtual desktop the window is in, however AWESOME window manager for instance does not do anything if the window is not on the current virtual desktop (this issue is currently discussed and a patch should be sent soon to fix this behavior).

It could also be possible to improve the slots allocation algorithm as it only adjusts the width of the window before being resized, however it should also include the width when the window is resized. However, this feature is not really important at the moment because it works fine at the moment.

4.6 Future plugins

This section lists the plugin I had planned to implement at first but I lacked time because the project was much more difficult than expected. However, by implementing the plugins above, I now have precise ideas about their implementations.

Some plugins would rely on mechanisms already explained in EXPOSÉ plugin such as downscaling algorithm and to get the `Pixmap` of *unmapped* windows.

4.6.1 Desktop switcher

4.6.1.1 Overview

It would allow to switch between desktops on the same screen by simply pressing a keyboard shortcut. This feature is common nowadays but a compositing manager would allow to provide live thumbnails of the desktop, thus improving considerably the usability of such feature. The figure 4.6 shows an example as implemented in `Kwin` [3].

[2] `http://mail.gnome.org/archives/wm-spec-list/2005-July/msg00029.html`

[3] `http://techbase.kde.org/Projects/KWin`

Figure 4.6: *Desktop switcher in* `Kwin`

4.6.1.2 Implementation

This plugin will provide thumbnails of each desktop and their windows and will be triggered either by a keyboard or mouse shortcut (set in the configuration file).

The plugin would require the window manager to implement the following `Atom`s, stored as root window properties, to get information about current desktops in use:

- **`_NET_NUMBER_OF_DESKTOPS`** to get the actual number of desktops on the screen;
- **`_NET_DESKTOP_GEOMETRY`** to get the common size of all desktops which will then be downscaled to allow all desktops to fit the screen;
- **`_NET_CURRENT_DESKTOP`** to get the current desktop which will be visually emphasized compared to other desktops previews. In addition, this `Atom` is needed when

switching to the desktop because a `ClientMessage` request is sent to the window manager which will switch to the wanted desktop;

- **`_NET_CLIENT_LIST`** to get the windows XID managed by the window manager. However, this list does not include windows created by the window manager such as toolbars and docs which will be obtained by getting the windows type as explained below;
- **`_NET_DESKTOP_NAMES`** to get the desktop names and display it along with each desktop. This atom is not actually required but would enhance usability from a user point of view.

In addition, in order to display thumbnails of windows on each desktop, it also requires the window manager to implement the following atoms stored as window properties:

- **`_NET_WM_DESKTOP`** to know on which desktop a window is currently in.
- **`_NET_WM_WINDOW_TYPE`** would allow to determine which windows should be displayed on all desktop (such as toolbars defined as `_NET_WM_WINDOW_TYPE_TOOLBAR` and docks defined as `_NET_WM_WINDOW_TYPE_DOCK`) among normal windows defined as `_NET_WM_WINDOW_TYPE_NORMAL`.

Like EXPOSÉ plugin, a new list of windows objects will be given to the rendering backend thanks to `render_windows` plugin hook. Basically, for a given desktop, the first window will be the desktop background, the next windows will be normal windows ordered by their stack position and at the end the docks and toolbars generally kept on top of all other windows. This way, the windows will be painted from the bottom to the topmost window of a virtual desktop.

4.6.1.3 Design refinement

This design will have to be refined in order to provide a preview of virtual desktops looking exactly like the window manager by taking into consideration the windows state as defined by `_NET_WM_STATE` window property.

In addition, EWMH specification does not state clearly how multiple monitor should be managed by the window manager, namely there is no specific atom to specify on which monitor

a desktop is currently in. Therefore, EWMH should be extended by providing an atom, stored as a root window property, to specify the number of monitors managed by the window manager along with its associated virtual desktops.

4.6.2 Application switcher

4.6.2.1 Overview

This effect is equivalent to `Exposé` except that it displays the list of windows of a specific virtual desktop. This feature already exists for ages but compositing manager would allow live thumbnails, thus improving enhancing usability of such feature from an user point of view. The figure 4.7 shows an example as implemented in `Compiz Fusion Switcher` [4] plugin.

Figure 4.7: *Application switcher in* `Compiz Fusion`

4.6.2.2 Implementation

This feature will provide thumbnails of each windows currently mapped on the current desktop and will be triggered by a keyboard or mouse shortcut set up in the configuration file.

First of all, this plugin would have to determine whose desktop the user is currently in by getting the **`_NET_CURRENT_DESKTOP`** root window property.

Secondly, it would require the window manager to implement the following atoms, stored as window properties:

[4] `http://wiki.compiz-fusion.org/Plugins/Switcher`

- **_NET_CLIENT_LIST** to get the windows managed by the window manager;
- **_NET_WM_DESKTOP** to select only windows on the current desktop;
- **_NET_ACTIVE_WINDOW** gives the currently active window;
- **_NET_WM_NAME** to display the window title below or above the windows thumbnail. This atom is not required but would allow the user to find out the wanted windows more easily.

Like EXPOSÉ plugin, a list of windows will be added to the current list of windows which will then be given to the rendering backend thanks to `render_windows` plugin hook. This list will contain thumbnails of the windows ordered according to their positing in the stack arranged from the current active window.

4.6.3 Screen magnifier

4.6.3.1 Overview

This accessibility plugin would allow to zoom in a portion of the screen around the mouse cursor. This might be particularly suitable for visually impaired people with some functional vision. The figure 4.8 shows an example as implemented in `Compiz Fusion Mag` [5] plugin.

4.6.3.2 Implementation

This feature may be triggered by pressing either a keyboard or mouse shortcut to enable it. The user may work on the portion on the screen zoomed in and then zoom out by using a keyboard shortcut. It requires no EWMH atom as it is just about zooming in the area under the mouse cursor by relying on `MotionNotify` mouse event generated by the X server when the cursor is moved around the screen.

Therefore it would just mean adding a window object to the end of the existing list of windows objects in order to paint it on top of existing windows. This new window object would be

[5] `http://wiki.compiz-fusion.org/Plugins/Mag`

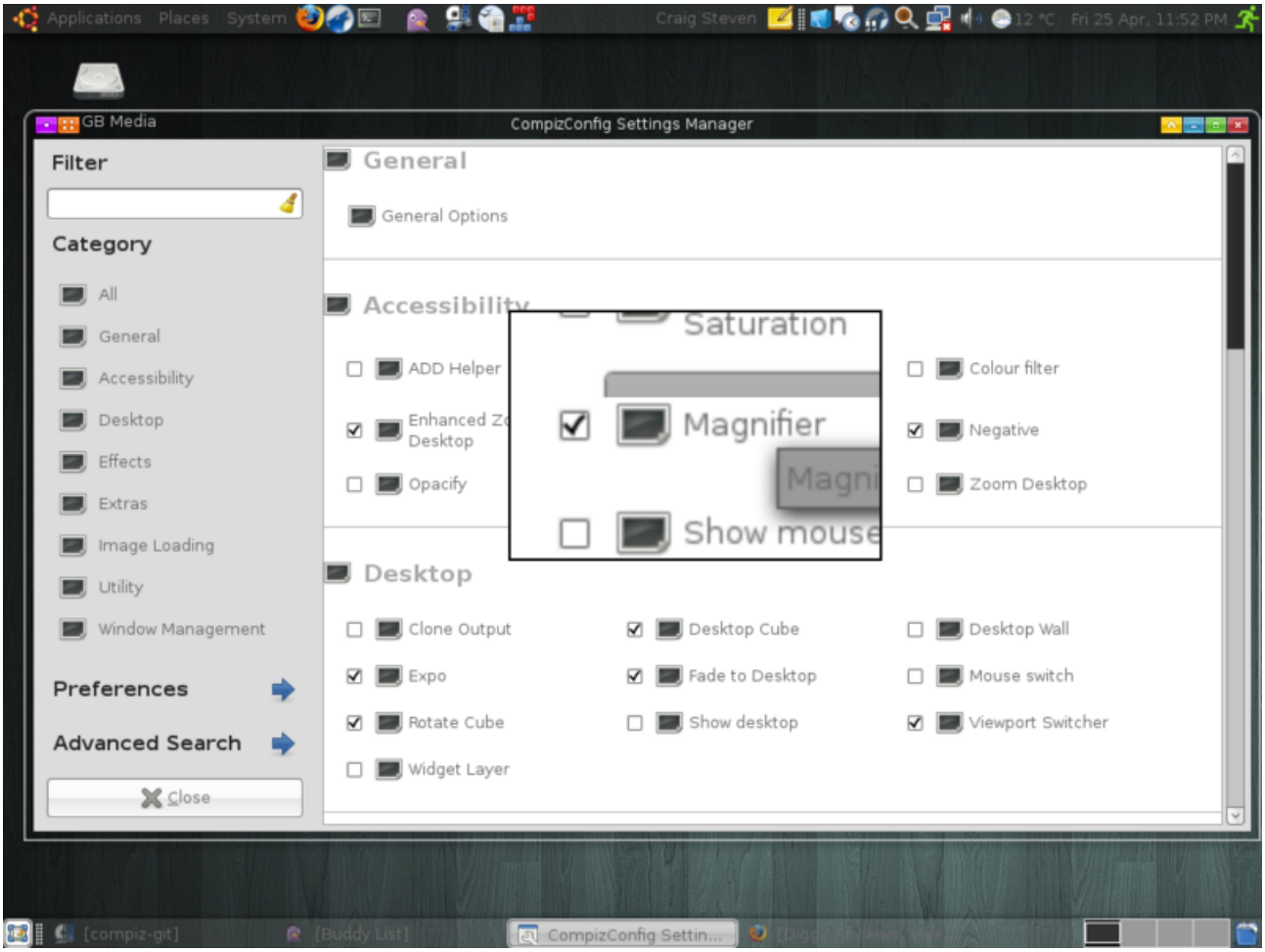

Figure 4.8: *Screen magnifier in* `Compiz Fusion`

a zoom of the region under the cursor which would be implemented based on the same mechanisms as EXPOSÉ (except that the image would be upscaling instead of being downscaling), namely `Image` as described in further details in section A.2.1.5.

Conclusion

At the moment, `xcb-ewmh` library has been completed and will be committed soon in XCB-UTIL GIT repository as it has been heavily tested and refined while writing the project code along with XCB extensions. Concerning the project, all the global architecture has been implemented so far, including the core code, rendering backend and effect plugins. Therefore, it will be easy from now to improve overall performances and add new features. Unfortunately, the software is not in a releasing shape yet because of performance issues which will be addressed soon though. At the beginning, I planned to implement all the plugins listed in the chapter 4, however it was already really hard and tricky to implement OPACITY and especially EXPOSÉ plugins. Because even though I was able to find easily documentations about X in general and I got help about that, there is not a lot of documentations, apart the protocol descriptions, about writing a compositing manager (and especially about X extensions involved) nor I did not get so much help on mailing lists and discussion channels. I also preferred to implement a neat architecture which could be easily extended rather than rushing on implementing plugins.

Thanks to this project, I learnt a lot about X programming and more generally about computer graphics because I did not know a lot about this area before. Even though it was really hard for me because of all the concepts to learn and understand, it was a really interesting and exciting experience from a technical and personal point of view. Moreover, some people are interested in using this project and even contributing, especially people coming from popular AWESOME window manager. Therefore, I am really motivated in continuing the development of this program, especially because I plan to use it on a daily basis along with AWESOME window manager.

I have already a lot of ideas about features to implement in the future, however I will firstly

focus on performance improvement before anything else. I am also very interested in experimenting OPENGL and especially OPENGL ES as the latter is currently being implemented in more and more embedded devices, such as mobile phones.

Appendix A

Appendix

A.1 Alpha compositing

Alpha compositing (Wikipedia 2009-04-20), also known as alpha blending or opacity, is about compositing two overlay images in a single one with partial transparency for overlapping areas, using a compositing algebra. Such effect can be achieved by adding an alpha channel to a pixel normally expressed by RGB channels and is referred as ARGB. An alpha channel is added to TRUECOLOR visual, the most common graphic method for decomposing a pixel value into RGB channels, where each channel would be 8 bits long, thus a pixel would be 32 bits long and allows approximately 16 million of colors.

The figure A.1 (Wikipedia 2009-04-20) shows alpha compositing on two overlapping images A and B using operators defined by Porter and Duff. A compositing manager only uses the *over* operator. This operator can be applied to the RGB channels of these images with their alpha channels, respectively c_a and c_b with α_a and α_b, by the following formula:

$$c = \frac{c_a * \alpha_a + c_b * \alpha_b(1-\alpha)}{\alpha_a + \alpha_b(1-\alpha_a)}$$

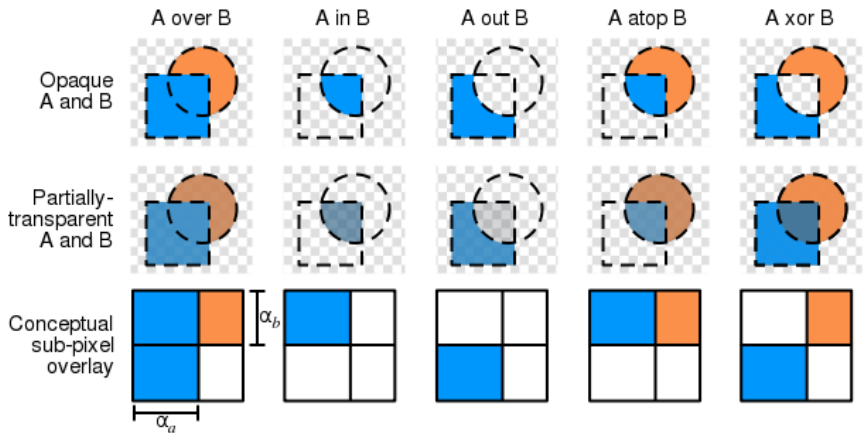

Figure A.1: *Porter and Duff compositing operator (Wikipedia)*

A.2 Libraries used

This project relies on several libraries, mainly from XCB (already described in section 1.2.2) and XCB-UTIL, therefore it is important to introduce them briefly.

A.2.1 XCB-UTIL libraries

As explained in section 1.2.2.2, XCB-UTIL libraries [1] intend to provide helpers not provided by XCB itself but in XLIB as they are not part of the X WINDOW PROTOCOL.

A.2.1.1 xcb-aux

This library provides *convenient access to connection setup and some core requests*. Among all the functions define in this library, only `xcb_aux_sync()` is used and allows to flush all pending requests to the X SERVER and blocks until all the requests have been processed. This function should be used when there is no other solution as blocking is costly and in most cases it can be avoided.

[1] `http://xcb.freedesktop.org/XcbUtil/`

A.2.1.2 xcb-event

This library intends to provide events handling functions through callbacks. It allows to define a callback for each X event or error as well as functions to process them in the queue. It also provides helpers to get the error and request labels but only for core protocol requests at the moment.

A.2.1.3 xcb-ewmh

This library has been written as part of this project because XCB lacks such library (XLIB does not implement EWMH either) and completely supports EWMH. As EWMH is getting more and more popular among window managers, it is worth implementing it. This library provides initialization of EWMH atoms and functions to manipulate them.

It uses M4 macro processing to avoid duplication of codes for atoms, and especially tedious maintainership because there are a lot of `Atoms` defined in EWMH. It also relies heavily on macro which may seem not easy to read at a first glance, however it is much better this way as it avoids duplicating the code at the cost of making errors reported by the compiler a bit harder to figure out.

A.2.1.4 xcb-icccm

This library provides an implementation of ICCCM specification which already exists in XLIB. It provides all the necessary helpers to manipulate ICCCM specific properties and mechanisms. However, this library has not been so used through this project because the windows properties defined in this specification have been deprecated by EWMH specification. Rather, some mechanisms, such as ownership, described in ICCCM are still relevant nowadays.

A.2.1.5 xcb-image

The X WINDOW PROTOCOL defines `PutImage` and `GetImage` requests which, respectively, combines an image given as a list of bytes with a rectangle of the drawable at the given coordinates and returns the image of the rectangle within a drawable. Both requests require

a format to be specified which might be either `XYPixmap` or `ZPixmap` where the former is organized as a set of bitmaps whereas the latter is organized as a set of pixels values.

The X WINDOW PROTOCOL only defines mechanisms, consequently, it is the X client library responsibility to provide helpers to manipulate easily an `Image`, for example, it could be interesting to be able to copy selected pixels to or from an `Image` according to the format. In XLIB, the implementation is known as `XImage`, nonetheless, XCB has lacked such implementation for a while, but it has recently been implemented as XCB-IMAGE. This library is particularly meaningful with RENDER backend to manipulate the windows contents.

A.2.1.6 xcb-keysyms

This library provides standard X key constants and functions to translate a `KeyCode` to a `KeySym` and vice versa. This is really meaningful because `KeyPress` and `KeyRelease` events contains a `KeyCode` which is then translated to a `KeySym` as detailed in the X WINDOW PROTOCOL manual.

However, this library is quite restricted because it does not contain functions to translate a `KeySym` to and from a string, like `XLookupString()` and `XStringToKeysym()` XLIB functions do, which is pretty useful when receiving a `KeyPress` event and comparing it to the keys defined in a configuration file for example. AWESOME project has implemented its own function to achieve that goal but this is not considered really clean because there is no XKB-like extension for XCB (hopefully someone is working on that at the moment and the work will probably be finished soon), therefore nothing has been integrated yet in this library and I currently have to re-use AWESOME code.

A.2.1.7 xcb-renderutil

This library defines, among others, helpers related to X RENDER extension, for instance to get the `PictFormat` associated with a `Visual`, meaningful when issuing a `CreatePicture` RENDER request for a given drawable. These helpers are not defined in the X WINDOW PROTOCOL, but implemented in XLIB.

A.2.2 libconfuse

libconfuse[2] library implements a simple but efficient configuration file parser written in C programming language which supports list of values and sections among other features.

A.2.3 libxdg-basedir

XDG freedesktop specification *defines where these files should be looked for by defining one or more base directories relative to which files should be located.* This specification is particularly meaningful for this project in order to store the configuration files in a standardized place (usually `/.config/` for per-user configuration files).

libxdg-basedir [3] implements an API to *list the directories according to the specification and provides a few higher-level functions*.

A.3 OPENGL libraries

A.3.1 GLITZ

GLITZ [4] intends to provide an efficient hardware accelerated 2D (Peter Nilsson June 27-July 2, 2004) OPENGL library which matches and extends the specification of the X RENDER extension, therefore it is a complete replacement of X RENDER extension. It intends to be used by an higher-level layer (such as CAIRO) which takes care of handling errors when performing a request not supported by the graphics hardware.

[2] `http://www.nongnu.org/confuse/`
[3] `http://n.ethz.ch/~nevillm/download/libxdg-basedir/`
[4] `http://freedesktop.org/wiki/Software/glitz`

A.3.2 CLUTTER

CLUTTER [5] is a modular OPENGL-based canvas library on which the window manager of the future GNOME 3 is built. It runs on a wide range of window system available on GNU/LINUX, WINDOWS and MAC OS X operating systems and implements both OPENGL and OPENGL ES (designed for embedded devices and implemented in SYMBIAN OS, ANDROID and IPHONE). It provides an XLIB back-end but not for XCB.

A.3.3 EVAS

EVAS [6], like CLUTTER, is an OPENGL-based canvas library part of EFL (ENLIGHTENMENT FOUNDATION LIBRARIES), mainly used by Enlightenment window manager [7]. Among its features, it can draw anti-aliased text and perform alpha blending in a hardware-accelerated way if possible, otherwise it fallbacks on X11 primitives.

A.4 Tools used

A.4.1 Building

Before building the program, the following libraries have to be installed beforehand:

- `XCB >= 1.4`;
  - `composite`;
  - `xfixes`;
  - `damage`;
  - `shape`;
  - `image`;

[5] `http://www.clutter-project.org/`
[6] `http://www.enlightenment.org/p.php?p=about/efl&l=en`
[7] `http://www.enlightenment.org/`

    - render.

- XCB-UTIL >= 1.5;
    - aux;
    - event;
    - ewmh;
    - renderutil;
    - keysyms.
- libxdg-basedir >= 1.0.0;
- libconfuse.

Further information about building the program and XCB-UTIL/ewmh library may be found in the INSTALL file provided in the source code tarball. In order to run the program, it is better at the moment to run it into XEPHYR (described in section A.4.2.3). Instructions about that are described in the README file provided in the source code tarball too.

A.4.1.1 Autotools

Autotools is part of the GNU build system and is a suite of programming tools designed to allow the building process portable on UNIX operating systems. It ships autoconf [8] (to produce configuration scripts to test operating systems capabilities and act accordingly), automake [9] (for creating portable Makefile) and libtool [10] (for creating portable libraries) tools widely known and popular across programmers on UNIX operating systems.

A.4.1.2 C programming language compiler

This program requires a C99-compliant C compiler as I am using features such as structures and variables initialization. Otherwise, it should not require any particular C compiler even if it

[8] http://www.gnu.org/software/autoconf/
[9] http://www.gnu.org/software/automake/
[10] http://www.gnu.org/software/libtool/

has only been tested with GCC [11] at the moment.

A.4.2 Debugging

This section describes the tools used when debugging the code while writing the code. GDB and `valgrind` are general tools commonly used when writing a program in C programming language, whereas `wireshark` and `Xephyr` are related to X debugging.

A.4.2.1 GDB

GDB [12] is a free software command line debugger, allowing for instance to set *breakpoints* on specific instructions and also display values of variables in memory. Along with XEPHYR, it allowed to debug (quite) easily synchronisation issues and more general bugs as well.

A.4.2.2 VALGRIND

VALGRIND [13] is a free software memory debugger and profiler. It allows to easily check whether the memory has been correctly freed and also detect errors such as read on invalid memory areas.

This tool was heavily used while writing the code to ensure there is no memory leak nor errors. Using this tool, I found a memory leak in libxdg-basedir [14] (section A.2.3).

A.4.2.3 XEPHYR

XEPHYR[15] is a nested X SERVER, meaning that it can run inside another X SERVER. This is particularly useful when trying to fix bugs in the code without switching to another X SERVER because it allows to debug it directly from the main X SERVER.

[11] `http://gcc.gnu.org/`
[12] `http://www.gnu.org/software/gdb/`
[13] `http://valgrind.org`
[14] `http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=539089`
[15] `http://www.freedesktop.org/wiki/Software/Xephyr`

A.5 Benchmarks of `PutImage` X request

The benchmark result A.1 and A.2 respectively shows the result of running `x11perf` [16] tool with and without SHM X extension.

```
$ x11perf -putimage500
x11perf - X11 performance program, version 1.2
The X.Org Foundation server version 10603000 on :0.0
from maggie
Thu Aug 20 03:56:02 2009

Sync time adjustment is 0.1452 msecs.

   1200 reps @   4.9776 msec (   201.0/sec): PutImage 500x500 square
   1200 reps @   5.1531 msec (   194.0/sec): PutImage 500x500 square
   1200 reps @   5.0596 msec (   198.0/sec): PutImage 500x500 square
   1200 reps @   5.0678 msec (   197.0/sec): PutImage 500x500 square
   1200 reps @   5.0198 msec (   199.0/sec): PutImage 500x500 square
   6000 trep @   5.0556 msec (   198.0/sec): PutImage 500x500 square
```

Listing A.1: *x11perf on* `PutImage` *without* SHM X *extension*

```
$ x11perf -shmput500
x11perf - X11 performance program, version 1.2
The X.Org Foundation server version 10603000 on :0.0
from maggie
Thu Aug 20 03:54:46 2009

Sync time adjustment is 0.1613 msecs.

  2400 reps @   2.2521 msec (   444.0/sec): ShmPutImage 500x500 square
  2400 reps @   2.4201 msec (   413.0/sec): ShmPutImage 500x500 square
  2400 reps @   2.2791 msec (   439.0/sec): ShmPutImage 500x500 square
  2400 reps @   2.3248 msec (   430.0/sec): ShmPutImage 500x500 square
  2400 reps @   2.2456 msec (   445.0/sec): ShmPutImage 500x500 square
 12000 trep @   2.3043 msec (   434.0/sec): ShmPutImage 500x500 square
```

Listing A.2: *x11perf on* `PutImage` *with* SHM X *extension*

[16] `http://cgit.freedesktop.org/xorg/app/x11perf/`

A.6 Source code

A.6.1 Source code organization

The program is split up among the following files. The source files are located in `src/` directory whereas the header files are in `include/` directory. The rendering backend and effect plugins are respectively in `rendering/` (chapter 3) and `plugins/` (chapter 4) directories.

A.6.1.1 `structs.h`

This header file defines the data types used all across the program. Rather, it defines `conf_t` data type to hold global variables such as the X connection, information about the screen (e.g. XID of the root window and its geometry), linked-list of windows and plugins and rendering backend related data.

A.6.1.2 `unagi.c`

This is the main source file and performs startup initialization, error handling and the main event loop. The startup is described in further details in section 2.3.2.

The main program accepts arguments to specify the path of the rendering backends and plugins libraries (respectively `rendering-path` and `plugins-path` options) and path of the configuration file (`config` option) parsed by using `getopt` from the C programming language standard library. It also parses the configuration file described in section A.6.2.

A.6.1.3 `util.c`

This source file provides miscellaneous helpers functions such as displaying error messages and will also contain a most efficient implementation of linked-lists as this data structure is widely used across the program.

A.6.1.4 `window.c`

It implements functions to manipulate the linked-list of windows among other general functions related to windows. The header file defines a new window data type (`window_t`) referred as a window object in this report.

A.6.1.5 `atoms.c`

It provides helpers related to `Atom`s, mainly initialization of EWMH, root background (stored usually in `_XROOTPMAP_ID` or `_XSETROOT_ID` root window properties) and opacity (`_NET_WM_WINDOW_OPACITY` opacity stored as a window property) `Atom`s.

A.6.1.6 `display.c`

This source file provides the following functions necessary to initialise a display and is only used during the startup of the program (the startup process is described in section 2.3.2).

A.6.1.7 `event.c`

It contains errors and events handling for both core and extensions requests. Each event handler calls all the associated plugins hooks if any and only if the plugin is enabled.

A.6.1.8 `plugin.c`

This file provides functions to load and unload all or one effect plugins in memory, whose architecture is described in chapter 4. Plus, it defines three data types, namely the plugin linked-list element (`plugin_t`), plugin virtual table (`plugin_vtable_t`) including events hooks (`plugin_events_notify_t`).

A.6.1.9 `rendering.c`

Similarly to `plugin.c`, this file provides functions to load and unload a rendering backend, whose architecture is described in chapter 3. It also defines a new data type, namely the

rendering backend virtual table (`rendering_t`).

A.6.2 Configuration file

The configuration file is parsed in the main source file, namely `unagi.c`, by using `libconfuse` library which has been retained because it is quite small, efficient and provide interesting features such as sections and list of values. At the moment, the configuration file is pretty short but may be expanded in the future to allow more configuration options (for example to set up the rendering backend in a more advanced way).

```
# Default rendering backend
rendering = "render"

# Plugins enabled
plugins = { "opacity", "expose" }
```

Listing A.3: *Configuration file syntax*

Each plugin can define its own section to support additional options (for instance the spacing between windows can be set up for EXPOSÉ effect plugin).

A.7 Performance comparison between XLIB and XCB

```
/*
 * Compilation: gcc -std=c99 `pkg-config --libs --cflags xcb` -lX11 \
 *                xwindow-xlib-xcb-performances.c -o \
 *                xwindow-xlib-xcb-performances
 *
 * This program is based on: http://xcb.freedesktop.org/tutorial/
 */

/* asprintf() */
#define _GNU_SOURCE

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>

#include <xcb/xcb.h>

#include <X11/Xlib.h>
```

```
21 static double
   get_time(void)
23 {
     struct timeval timev;
25   gettimeofday(&timev, NULL);

27   return (double) timev.tv_sec + (((double)timev.tv_usec) / 1000000);
   }
29
   int
31 main(void)
   {
33   unsigned int i;
     double start, end, diff;
35
     /* Number of IntermAtom X requests sent for performance
37      measurement */
     const int count = 500;
39
     /* Init names */
41   char *names[count];
     for(i = 0; i < count; ++i)
43       asprintf(names + i, "NAME%d", i);

45   xcb_connection_t *conn = xcb_connect(NULL, NULL);

47   /* Beginning of XCB synchronous method (wrong usage!) */
     xcb_atom_t atoms[count];
49   start = get_time();

51   for(i = 0; i < count; ++i)
       atoms[i] = xcb_intern_atom_reply(conn,
53                                        xcb_intern_atom(conn, 0,
                                                          strlen(names[i]),
55                                                        names[i]),
                                          NULL)->atom;
57
     end = get_time();
59
     printf("=>␣XCB␣wrong␣usage␣(synchronous)\n");
61   printf("Duration:␣%fs\n\n", end - start);
     diff = end - start;
63   /* End of XCB synchronous method */

65   /* Beginning of XCB asynchronous method (good usage!) */
     xcb_intern_atom_cookie_t atom_cookies[count];
67   xcb_atom_t atoms_xcb[count];
     start = get_time();
69
     /* Send the request */
71   for(i = 0; i < count; ++i)
       atom_cookies[i] = xcb_intern_atom(conn, 0, strlen(names[i]),
```

```
                                        names[i]);

  /* Now get the replies */
  for(i = 0; i < count; ++i)
    {
      xcb_intern_atom_reply_t *r =
        xcb_intern_atom_reply(conn,
                              atom_cookies[i],
                              0);

      if(r)
        atoms_xcb[i] = r->atom;

      free(r);
  }

  end = get_time ();

  printf("=> XCB bad usage (asynchronous)\n");
  printf("Duration: %fs\n", end - start);
  printf("Ratio    : %.2f\n\n", diff / (end - start));
  diff = end - start;
  /* End of XCB asynchronous method */

  /* Close XCB connection to the X server */
  xcb_disconnect(conn);

  /* Xlib method

     Note: one could argue that the library provides XinternAtoms()
           function which sends the atoms asynchronously, but
           asynchronous method is quite complicated to use in Xlib, in
           addition, this is not the purpose of this program */
  Display *dpy = XOpenDisplay(NULL);

  Atom atoms_xlib[count];
  start = get_time();

  for (i = 0; i < count; ++i)
    atoms_xlib[i] = XInternAtom(dpy, names[i], 0);

  end = get_time();

  printf("=> Xlib traditional usage\n");
  printf("Duration  : %fs\n", end - start);
  printf("Ratio     : %.2f\n", (end - start) / diff);
  /* End of Xlib method */

  /* Free variables */
  for (i = 0; i < count; ++i)
    free(names[i]);
```

```
125   /* Close the Xlib connection to the X server */
      XCloseDisplay(dpy);
127
      return EXIT_SUCCESS;
129 }
```

Listing A.4: *Performance comparison between* XLIB *and* XCB

A.8 Example of `visual` available

```
1  number of visuals:    8
   default visual id:  0x21
3  visual:
     visual id:    0x21
5    class:    TrueColor
     depth:    24 planes
7    available colormap entries:    256 per subfield
     red, green, blue masks:    0xff0000, 0xff00, 0xff
9    significant bits in color specification:    8 bits
   visual:
11   visual id:    0x22
     class:    DirectColor
13   depth:    24 planes
     available colormap entries:    256 per subfield
15   red, green, blue masks:    0xff0000, 0xff00, 0xff
     significant bits in color specification:    8 bits
17 visual:
     visual id:    0x6d
19   class:    TrueColor
     depth:    24 planes
21   available colormap entries:    256 per subfield
     red, green, blue masks:    0xff0000, 0xff00, 0xff
23   significant bits in color specification:    8 bits
   visual:
25   visual id:    0x6e
     class:    TrueColor
27   depth:    24 planes
     available colormap entries:    256 per subfield
29   red, green, blue masks:    0xff0000, 0xff00, 0xff
     significant bits in color specification:    8 bits
31 visual:
     visual id:    0x6f
33   class:    DirectColor
     depth:    24 planes
35   available colormap entries:    256 per subfield
     red, green, blue masks:    0xff0000, 0xff00, 0xff
37   significant bits in color specification:    8 bits
    visual:
39   visual id:    0x70
     class:    DirectColor
```

```
41    depth:    24 planes
      available colormap entries:    256 per subfield
43    red, green, blue masks:    0xff0000, 0xff00, 0xff
      significant bits in color specification:    8 bits
45   visual:
      visual id:    0x71
47    class:    DirectColor
      depth:    24 planes
49    available colormap entries:    256 per subfield
      red, green, blue masks:    0xff0000, 0xff00, 0xff
51    significant bits in color specification:    8 bits
    visual:
53    visual id:    0x64
      class:    TrueColor
55    depth:    32 planes
      available colormap entries:    256 per subfield
57    red, green, blue masks:    0xff0000, 0xff00, 0xff
      significant bits in color specification:    8 bits
```

Listing A.5: *Example of* `Visuals` *available (obtained by calling* `xdpyinfo` *command)*

Bibliography

Apple (2008-10-15), 'Max OS X Technology Overview: Graphics and Multimedia Technologies'. *last access: 2009-07-15*.

URL: *http://developer.apple.com/documentation/MacOSX/Conceptual/OSX_Technology_Overview/GraphicsTechnologies/GraphicsTechnologies.html*

consortium, X. (2005-10-31), 'The Xlib Manual: Visual Types'. *last access: 2009-08-23*.

URL: *http://tronche.com/gui/x/xlib/window/visual-types.html*

Corbet, J. (1991), 'MIT-SHM - The MIT Shared Memory Extension How the shared memory extension works'. *last access: 2009-08-19*.

URL: *http://www.xfree86.org/current/mit-shm.html*

Eich, E. (2004-04-23), 'XAA.HOWTO'. *last access: 2009-08-17*.

URL: *http://cgit.freedesktop.org/xorg/xserver/plain/hw/xfree86/xaa/XAA.HOWTO*

Fedora (2008-05-24), 'Architectures for a Compositing Manager'. *last access: 2009-08-16*.

URL: *http://fedoraproject.org/wiki/RenderingProject/CMArchitecture*

Group, K. (2009), 'EGL Overview - Native Platform Graphics Interface'. *last access: 2009-08-18*.

URL: *http://www.khronos.org/egl/*

Group, X. D. (2006-09-29), 'Extended Window Manager Hints'. *last access: 2009-08-14*.

URL: *http://standards.freedesktop.org/wm-spec/wm-spec-latest.html*

Hanemaayer, H. (1996), 'XAA.HOWTO'. *last access: 2009-08-23*.

URL: *http://cgit.freedesktop.org/xorg/xserver/plain/hw/xfree86/xaa/XAA.HOWTO*

Höglund, F. (n.d.), 'Composite tutorial'. *last access: 2009-08-09*.

URL: *http://ktown.kde.org/~fredrik/composite_howto.html*

Jackson, A. (2005-04-12), 'Development/Documentation/Performance'. *last access: 2009-08-18*.

URL: *http://www.x.org/wiki/Development/Documentation/Performance*

Jackson, A. (2005-08-21), 'ExaStatus'. *last access: 2009-08-12*.

URL: *http://www.x.org/wiki/ExaStatus*

Keith Packard, D. J. (2007-07-03), 'Composite Extension'. *last access: 2009-08-22*.

URL: *http://cgit.freedesktop.org/xorg/proto/compositeproto/plain/compositeproto.txt*

Keith Packard, E. A. (2007-01-08), 'The DAMAGE Extension'. *last access: 2009-08-22*.

URL: *http://cgit.freedesktop.org/xorg/proto/damageproto/plain/damageproto.txt*

Keith Packard, J. G. (2000-06-15), 'The (Re)Architecture of the X Window System'. *last access: 2009-08-22*.

URL: *http://keithp.com/~keithp/talks/xarch_ols2004/xarch-ols2004-html/*

Leech, J. (n.d.), 'OpenGL Graphics with the X Window System (Version 1.4)'. *last access: 2009-08-14*.

URL: *http://www.opengl.org/documentation/specs/glx/glx1.4.pdf*

Packard, K. (2000-04-19), 'A New Rendering Model for X'. *last access: 2009-08-22*.

URL: *http://www.keithp.com/~keithp/talks/usenix2000/render.html*

Packard, K. (2005-07-01), 'The X Rendering Extension'. *last access: 2009-08-22*.

URL: *http://cgit.freedesktop.org/xorg/proto/renderproto/plain/renderproto.txt*

Packard, K. (2006-12-14), 'The XFIXES Extension'. *last access: 2009-08-22*.

URL: *http://cgit.freedesktop.org/xorg/proto/fixesproto/plain/fixesproto.txt*

Peter Nilsson, D. R. (June 27July 2, 2004), Glitz: Hardware Accelerated Image Compositing using OpenGL, *in* 'Proceedings of the FREENIX Track: 2004 USENIX Annual Technical Conference', USENIX Association, Boston, MA, USA. last access: 2009-08-16.

URL: *http://www.usenix.org/events/usenix04/tech/freenix/nilsson.html*

Salminen, R. (2009-06-26), 'XCB and OpenGL documentation draft'. *last access: 2009-08-17*.

URL: *http://kesakoodi2k9.wordpress.com/2009/06/26/xcb-and-opengl-documentation-draft/*

Siracusa, J. (2005-04-28), 'Mac OS X 10.4 Tiger: Quartz'. *last access: 2009-08-15*.

URL: *http://arstechnica.com/apple/reviews/2005/04/macosx-10-4.ars/13*

Smirl, J. (2005-08-30), 'The State of Linux Graphics'. *last access: 2009-08-18*.

URL: *http://jonsmirl.googlepages.com/graphics.html*

Standard, X. C. (1993-12), 'Inter-Client Communication Conventions Manual'. *last access: 2009-08-23*.

URL: *http://www.x.org/docs/ICCCM/icccm.pdf*

Wikipedia (2009-01-28), 'Quartz Compositor'. *last access: 2009-08-19*.

URL: *http://en.wikipedia.org/wiki/Quartz_Compositor*

Wikipedia (2009-02-19), 'Desktop Window Manager'. *last access: 2009-08-02*.

URL: *http://en.wikipedia.org/wiki/Desktop_Window_Manager*

Wikipedia (2009-02-24), 'X Window System protocols and architecture'. *last access: 2009-07-23*.

URL: *http://en.wikipedia.org/wiki/X_Window_System_protocols_and_architecture*

Wikipedia (2009-02-25), 'Compositing window manager'. *last access: 2009-07-27*.

URL: *http://en.wikipedia.org/wiki/Compositing_window_manager*

Wikipedia (2009-03-09*a*), 'Compiz'. *last access: 2009-08-23*.

URL: *http://en.wikipedia.org/wiki/Compiz*

Wikipedia (2009-03-09*b*), 'Exposé'. *last access: 2009-07-16*.

URL: *http://en.wikipedia.org/wiki/Expose_(Mac_OS_X)*

Wikipedia (2009-04-20), 'Alpha compositing'. *last access: 2009-08-10*.

URL: *http://en.wikipedia.org/wiki/Alpha_compositing*

Wikipedia (2009-06-07), 'Gaussian blur'. *last access: 2009-08-15*.

URL: *http://en.wikipedia.org/wiki/Gaussian_blur*

Worth, C. (2009-02-19), 'Running render_bench against EXA/i965'. *last access: 2009-08-17*.

URL: *http://cworth.org/exa/i965/render_bench/*

Worth, C. (2009-06-10), 'A first look at Glucose on the i965'. *last access: 2009-08-12*.

URL: *http://www.cworth.org/glucose/i965/first_look/*

X Consortium Standard, R. W. S. (2004), 'X Window System Protocol'. *last access: 2009-07-15*.

URL: *ftp://ftp.x.org/pub/X11R7.0/doc/PDF/proto.pdf*

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to

text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent

copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of

Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated

and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.